VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Využití frameworku Tapestry pro vývoj moderních webových aplikací
Utilization of Tapestry framework for modern web application development

2010 Ondřej Dočkal

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. 5. 2010 . . . . . .

Poděkování patří všem, kteří jakkoliv ovlivnili moji práci, pomáhali poznámkami nebo radami a přispívali tak k jejímu dokončení. Poděkování tedy patří rodině za podporu při studiu, mému vedoucímu p. Ing. Filipcovi za poskytnuté poznámky k mojí práci a spolužákům a přátelům, kteří v tom byli se mnou.

Abstrakt a klíčová slova

Abstrakt:

Cílem této práce je zhodnotit využití webového aplikačního frameworku Tapestry v návrhu a implementaci moderních webových aplikací s ohledem na jejich požadavky. Práce je rozdělena do šesti kapitol. V první kapitole je úvod do problematiky webových aplikací, architektonických vzorů a frameworků. Poté se práce zabývá požadavky na moderní webové aplikace. Následují Java technologie a typické rysy Java frameworků a jejich srovnání. Jak paradigmaty moderních webových aplikací řeší Tapestry je pospáno v další části. Postupně je rozebráno jak přesně Tapestry implementuje dané problémy a jaké jsou jeho základní rysy. Nakonec jsou nastíněny postupy tvorby a implementace některých částí ukázkové aplikace. Vše je zhodnoceno v závěru.

Klíčová slova:

Tapestry, Požadavky moderních webových aplikací, Tapestry 5, Model-View-Controller, webový aplikační framework,

Abstract and key words

Abstract:

The aim of this study is the utilization of a Tapestry web application framework in the design and implementation of modern web applications with regard to their requirements. The thesis is divided into six chapters. The first chapter is an introduction into web applications, architectural patterns and frameworks. Then the work deals with the requirements of modern web applications. The following are Java technologies and the typical features of Java frameworks and their comparison. How the paradigms of modern web applications are solved by Tapestry is described in the next section. Gradually we discuss how exactly Tapestry implements the problems and what are its basic features. Finally, procedures are outlined for the development and implementation of certain parts of the sample applications. Everything is evaluated at the end.

Key words:

Tapestry, requirements of modern web applications, Tapestry 5, Model-View-Controller, web application framework

Seznam použitých zkratek a symbolů

(X)HTML – (Extensible)HyperText Markup Language

XML – Extensible Markup Language

URL – Uniform Resources Locator

MVC – Model – View – Controller

MVP – Model – View - Presenter

UI – User Interface

JDBC – Java DataBase Connector

HTTP(S) – (Secured) HyperText Transfer Protocol

SEO – Seach Engine Optimization

API – Application Programming Interface

SŘBD – Systém Řízení Báze Dat

SQL – Structured Query Language

i18n – Internationalization

l10n – Localization

AJAX – Asynchronous JavaScript and XML

DOM – Document Object Model

JSP – JavaServer Pages

JEE – Java Enterprise Edition

ORM – Objected Relational Mapping

POJO – Plain Old Java Object

JSTL – JavaServer Pages Standard Tag Library

EJB – Enterprise Java Bean

JDO – Java Data Object

IoC – Inversion of Control

EL – Expression Language

JPA – Java Persistence API

TML – Tapestry Markup Language

IDE – Integrated Development Environment

JDK – Java Development Kit

Obsah

1. Úvod 1
2. Nástroje pro tvorbu webových aplikací 2
   2.1 Webová aplikace 2
   2.2 Návrhové a architektonické vzory 3
   2.3 Architektonický vzor MVC 5
   2.4 Framework 9
   2.5 Shrnutí 12
3. Požadavky na moderní webové aplikace 13
   3.1 Vnitřní funkční požadavky aplikace 13
   3.2 Vnější funkční požadavky 15
4. Srovnání Java Frameworků 16
   4.1 Java technologie 16
   4.2 Přehled 16
   4.3 Základní rysy 17
   4.4 Popis webových aplikačních frameworků 18
   4.5 Srovnání 20
5. Tapestry 5 v moderních aplikacích 23
   5.1 Historie 23
   5.2 Zdroje 23
   5.3 Využití Tapestry 5 v moderních webových aplikacích 24
   5.4 Shrnutí 26
6. Základní rysy Tapestry5 27
   6.1 Pracovní prostředí 27
   6.2 Struktura projektu Tapestry 28
   6.3 Zpracování požadavku 30
   6.4 Komponenty, stránky a šablony 31
   6.5 Anotace 36
   6.6 Události komponent 36
   6.7 Lokalizace 38
   6.8 Live class and template reloading 38
   6.9 Tapestry služby 39
   6.10 Shrnutí 39
7. Tapestry5 aplikace 40
   7.1 Navigace 40
   7.2 Hlášení chyb 41
   7.3 Application state object, konfigurace, ...41
   7.4 Komponenty Grid a BeanEditForm 43
   7.5 Validace 44
   7.6 Implementace MVC 45
8. Závěr 47

1. Úvod

Moderní doba nám nabízí neocenitelný poklad – informace přístupné v takové míře, o které se nám před dvěma desetiletími ani nesnilo.

Všechny tyto informace a služby poskytované v prostoru zvaném Internet mají jedno společné – všechny byly vytvořeny člověkem – programátorem, developerem, designérem nebo prostým amatérem. A jsou to právě webové aplikace, které poskytují informace a služby dohromady. Hlavním tématem této práce je právě paradigma jejich tvorby.

Na samém začátku je úvod do problematiky webových aplikací, co to vlastně tyto aplikace jsou. Poté následuje vysvětlení rodílů mezi návrhovými vzory, které se zabývají řešením obecných problémů a architektonickými vzory, které již slouží jako podklad k budování dané struktury aplikace.

Některé nástroje velmi ulehčují práci vývojářům tím, že implementují určitou problematiku funkcionality aplikací. Jedná se o frameworky, především se pak zabývám Java frameworky a jejich využitím a srovnáním na základě požadavků moderních webových aplikací.

A jak se se všemi těmito okruhy a problematikou vyrovnává Java framework Tapestry je rozebráno v dalších kapitolách. První na řadu příjde použití Tapestry v ohledu na požadavky moderních aplikací.

Základní rysy Tapestry 5 demonstrované na příkladech, ukázky použití klasické funckionality a popis většiny vlastností, které framework nabízí, jsou rozebrány v předposlední části práce. Poslední část se zabývá konkrétní implementací částí ukázkové aplikace a shrnutím funckionality a architektury frameworku.

2. Nástroje pro tvorbu webových aplikací

Pro začátek bych zmínil, že některé definice a pojmy jsem uváděl podle [1] a [2] zdrojů, které zde fungují, jako de-facto prostředník k uznávaným článkům, standardům a norem, ze kterých samy čerpají.

2.1 Webová aplikace

V oblasti softwarového inženýrství se webovou aplikací myslí taková aplikace, která je poskytovaná uživatelům z webového serveru přes síť Internet, popřípadně v rámci firemní či korporátní sítě – Intranet. Tyto aplikace využívají webový prohlížeč jako tzv. tenkého klienta, což znamená, že veškerá funkcionalita a aplikační logika se nachází na straně serveru anebo se software může vyskytovat jako javovský applet či využívat prohlížečem podporovaný jazyk JavaScript (se značkovacím jazykem (X)HTML či XML pro generování stránek)[1]. Komunikace mezi serverem a klientem probíhá prostřednictvím protokolu HTTP.

Nespornou výhodou je to, že většina subjektů v kybernetickém prostoru využívá právě prohlížeč jako zobrazovací médium dat přicházejících z Internetu, a proto jsou webové aplikace tolik rožšířené a používané. Nejvyužívanějšími klientskými webovými browsery jsou Internet Explorer, Mozzila Firefox, Google Chrome, Opera anebo Safari.

Další výhodou je možnost správy a aktualizace aplikace bez nutnosti šířit software či jeho update na všechny uživatelské entity. Specifické implementace tohoto druhu softwaru jsou především využívány jako podnikové systémy, informační systémy či jako freemaily, e-shopy, aukce, fóra apod[1].

2.1.1 Specifikace webové aplikace

2.1.1.1 Statický web

Jsou systémy, ve kterých se jako vstupní parametr k lokalizaci dokumentu používá URL. Ten přímo odkazuje na statický dokument, jenž drží odkazy na další dokumenty. Zde prakticky neexistuje potřeba nějakého složitého návrhu webu.

2.1.1.2 Dynamická aplikace

Jedná se o aplikace, které zobrazují uživateli data (stránky pomocí (X)HTML) v závislosti na jeho požadavcích odeslaných serveru. Starší aplikace využívaly program nainstalovaný na straně klienta, aby rozšířily funkcionalitu aplikace, to však s sebou přineslo komplikace v případech, kdy se program musel aktualizovat, což obnášelo distribuci nové části na všechny klientské stroje [1].

Později se začal používat skriptovací jazyk JavaScript dovolující tvůrci aplikace přidat dynamické prvky, které pracovaly na klientské straně, čímž přestaly být zasílány pouze statické HTML odpovědi ze serveru a obsah mohl být generován podle požadavků od uživatele.

2.1.1.3 Struktura webové aplikace

Aplikace jsou většinou rozděleny na logické části – vrstvy. Každá vrstva má svou roli. Mnoho

aplikací sestává pouze z jedné vrstvy, jenomže vývoj je přirozeně škatulkuje do více vrstev. Nejčastější strukturou je třívrstvá architektura. Každá vrstva se nazývá podle své role: Prezenční, aplikační a úložná. Prezentační vrstvu zde může zastupovat webový prohlížeč. Aplikační vrstva je tvořena jedním z nástrojů, které tvoří Webovou aplikaci (ASP. NET, JSP/JAVA, PHP, Python, Ruby on Rails nebo Struts) a implementuje tzv. aplikační logiku. A poslední vrstvou je úložná vrstva nebo-li model a tvoří ji většinou uložiště dat – databáze a její interpretace v aplikaci[1].

Princip spočívá v postupu, kdy prohlížeč pošle požadavek do aplikační vrstvy, která zajistí vytvoření či provedení dotazu do databáze (načtení, uložení, změnu anebo smazání objektu), a vygeneruje uživatelské rozhraní[1].

Samozřejmě toto není jediná možnost strukturování, tvorby a vůbec fungování webové aplikace, ale pro přiblížení k jejímu pochopení postačí. Dalším krokem bude náhled do návrhových vzorů.

2.2 Návrhové a architektonické vzory

Každá webová aplikace představuje pro vývojáře sadu otázek a problémů, svázanou s jejími požadavky.

Nároky na aplikaci, ač už se jedná o soukromého zadavatele nebo projekt firmy, hodnotí a pokryjí softwaroví analytici. Na této úrovni se řeší otázky typu: „Pro koho je aplikace určena.“, „Jaké služby bude nabízet.“ popř. „Jedná se o Open source anebo o vlastnický software?“. Tato analýza se dále skládá z jednotlivých části:

- Statický pohled na aplikaci zahrnuje konceptuální modelování (ER model, UML).
- Funkce systému zahrnuje analýza procesů (případy použití, identifikace procesů, seznam procesů, Data flow a sekvenční diagramy), vazby mezi objekty aplikace a datové toky.

2.2.1 Návrhové vzory

Pokud je již zrealizována analýza, přichází na řadu skutečné rozhodování, jak se k implementaci sktruktury aplikace postavit. Nemám na mysli pouze rozdělení aplikace do několika vcelku přehledných vrstev, tedy její architekturu, ale také přímé provádění jednotlivých procesů uvnitř systému: „Jak správně autentizovat a autorizovat uživatele aplikace.“, „Obsluhování jednotlivých procesů ovlivňující uložená data, po zadání požadavků uživatele.“ nebo „Zobrazení požadovaných dat ve webovém prohlížeči.“

„Z těchto důvodů vznikl jakýsi jazyk, jako již jednou osvědčené řešení problémů zachytit pro další vývojáře. Začaly vznikat návrhové vzory (Design patterns), které si daly za úkol popisy určitých problémů zachytit v systematicky uspořádané formě. Není účelem postihnout ve struktuře návrhových vzorů všechny možné problémy. Dobrý vzor musí představovat velmi obecné řešení netriviálního problému, který se často vyskytuje. Musí mít ustálenou terminologii a být někde implementován.“[4]

Zde můžeme využít již vytvořených návrhových vzorů řešících převážně tyto obecné problémy, vytvořené na základě zkušeností jiných vývojářů. Poznamenávám, že nezávisí na implementačním prostředí, ani na použitém programovacím jazyce. Dá se říct, že pokud použijeme návrhový vzor v již popsané situaci, měli bychom dosáhnout řešení.

2.2.1.1 Základní návrhové vzory

Základní rozdělení návrhových vzorů je do tří skupin[5].

- *Creational Patterns* (vytvářející vzory)
- *Structural Patterns* (strukturální vzory)
- *Behavioral Patterns* (vzory chování)
- *Concurrent patterns* (souběžné vzory)

2.2.1.2 Creational patterns

Při vytváření objektů v systému mohou nastat problémy, které řeší právě tyto návrhové vzory. Jejich cílem je popsat správný postup výběru objektu dané třídy a také ošetřující výběr vhodného počtu objektů. Vytváření instantí tříd většinou probíhá dynamicky za běhu programu. (*Factory Method Pattern, Abstract Factory Method Pattern, Builder Pattern, Prototype Pattern, Singleton Pattern*).

2.2.1.3 Structural patterns

Tyto návrhové vzory poskytují členící možnosti prvků v systému, převážně tříd a komponent. Toto počíná by mělo zpřehlednit systém a strukturalizovat kód. (*Adapter Pattern, Bridge Pattern, Composite Pattern, Decorator Pattern, Facade Pattern, Flyweight Pattern, Proxy Pattern*).

2.2.1.4 Behavioral patterns

Jak už název napovídá, behaviorální návrhové vzory se specializují na ošetření chování prvků v systému, především na třídy a jejich objekty. Využívají principy dědičnosti a dále popisují spolupráce a vztahy mezi objekty a jejich skupinami. (*Chain Of Responsibility Pattern, Command Pattern, Interpreter Pattern, Iterator Pattern, Mediator Pattern, Memento Pattern, Observer Pattern, State Pattern, Strategy Pattern, Template Pattern , Visitor Pattern*).

2.2.1.5 Concurrent patterns

Tyto vzory se zabývají problémy vzniklými spouštěním programů ve vláknech. Tedy takovými problémy, které vznikají při paralelním řešením instrukcí nebo úloh.

2.2.2 Architektonické vzory

Architektonický vzor vyjadřuje základní strukturální organizaci nebo schéma softwarového systému. Poskytuje soubor předdefinovaných podsystémů, specifikujících jejich zodpovědnosti, pravidla a postupy pro definování vztahů mezi nimi[6].

Rozdíly mezi jednotlivými architektonickými vzory spočívají v zaměření kvality výsledné aplikace, například na výkon či větší dostupnost pro uživatele. Právě tyto rozhodnutí by měl vývojář vzít v potaz před samotnou implementací, protože každý arch. vzor poskytuje zvýhodnění jiné části aplikace. Rozdělení klasických vzorů softwarových architekrur je následující:[3]

- *Vrstvový vzor* – popisuje architekturu, kde může být systém jako heterogenní entita rozložen do více navzájem komunikujících částí.

- *Vzor datový tok* – popisuje architekturu, kde mohou být toky dat úspěšně zpracovány nebo tranformovány komponentami systému.
- *Data centrický vzor* – je vhodný pokud je v systému zakomponován centrální repositář, který je zpřístupněn více komponentám najednou.
- *Adaptační vzo*r – popisuje adaptaci systému při jeho samotném vývoji.
- *Vzor jazykového rozšíření* – se zabývá, jakým způsobem systém nabízí abstrakční vrstvu početní infrastruktuře.
- *Vzor uživatelské interakce* – nabízí komponentní strukturu programu s uživatelským rozhraním, pracující v reálném čase. Právě zde patří architekruta *Model – View – Controller*, kterou se budeme dále zabývat.
- *Vzor komponentní interakce* – soustředí se na to, jak se vyměňují zprávy mezi jednotlivými komponentami, aniž by ztratily své role.
- *Distribuční vzor* – řeší problémy s rozšiřováním komponent v síťovém prostředí.

Po zpřehlednění pojmů jako architektonické a návrhové vzory se můžeme začít zabývat architekturou MVC nebo-li Model – View – Controller (pokud bude uveden název s velkým písmenem, myslím tím komponentu MVC vzoru), která se používá ve většině aplikací. Nejenom, že tyto softwarové struktury tvoří nejrozličnější aplikace jak desktopové, tak webové, ale také jsou obsaženy v pracovních rámcích tzv. frameworcích, které pomáhají tvořit vysoce specializované aplikce ve všech programovacích jazycích.

2.3 Architektonický vzor MVC

Počátky architektury MVC (Model – View – Controller) spadají do konce 70. let, kde v jazyce Smalltalk bylo nutné ošetřit uživatelský vstup, zpracování požadavku a poté také výstup na uživatelské rozhraní.[7] Tento vzor se taktéž osvědčil v budování vícevrstvé aplikace, kde se využívá dodnes v různých variacích. S nástupem webových aplikací s ovládacími prvky se přesunula některá aplikační logika z Controlleru na View a vzor MVC by již nebyl přesný, proto vznikly její variace Model – View – Presenter (MVP)[8].

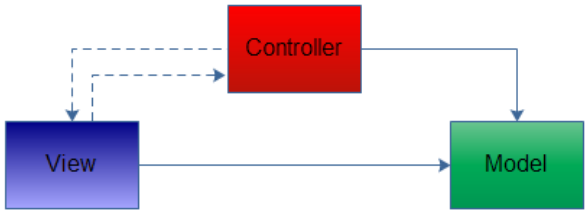
2.3.1 Základní rozbor

Obecně dělí architektura MVC aplikaci na 3 logické části: Model (model), View (pohled) a Controller (řadič). V práci budu užívat anglické názvosloví, jelikož má jasnější význam než české překlady.

Každá část má jiný úkol, a také by se měla chovat tak, že při změně jedné z nich to ostatní příliš neovlivní. Model reprezentuje data a datovou logiku, View se stará o jejich zobrazování v uživatelském rozhraní a Controller ošetřuje aplikační logiku a události z UI.

- Model – Jedná se o doménový model, který popisuje jak objekty reálného světa, tak i vztahy mezi nimi a jejich interakce. Zjednodušeně by se dal model považovat za datovou strukturu a její modelovou logiku. Nejedná se o skutečnou databázi, ale o její prezentaci. Pomocí databázového konektoru JDBC nebo ODBC lze propojit daný model se skutečnou databází. Při správném použití této architektury je model zcela zapouzdřen.

- View – Se stará o vykreslování dat z Modelu do uživatelského rozhraní společně s grafickými prvky. Při změně stavu Modelu View zažádá o nové informace a přepíše data v UI.
- Controller – Obstarává aplikační logiku aplikace, převážně se stará o převedení požadavku od uživatele na dotaz směřující a ovlivňující Model, není přímo jeho činitelem, ale spíše se jedná o prvek delegující procesy v modelové vrstvě.

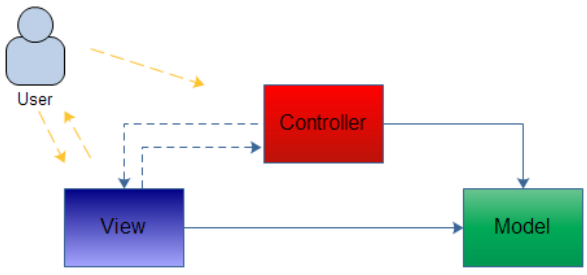


Obr. 2.1: Návaznost komponent architektury MVC

Na obrázku Obr. 2.1 vidíme vazby mezi prvky vzoru MVC. Plné čáry mají funkci přímých vazeb, které jsou zde podstatné. Je z nich patrné, že Model není přímým činitelem děje. Zato Controller má přímou vazbu do Modelu, kde spouští procesy spravující stav objektů. View zase využívá přístupu do Modelu, ale pouze kvůli načtení dat a jejich zobrazení nemá žádnou možnost ovlivnit data v Modelu. Žádné další vazby nejsou v tuto chvíli podstatné. V dalších variacích této architektury se můžeme setkat s vazbami mezi Controllerem a View.

2.3.2 MVC v interakci s uživatelem

Abychom celkově pochopili architekturu softwarové aplikace, musíme ji postavit do situace, kde bude vcházet do vztahu s běžným uživatelem. To znamená, jak aplikace reaguje na uživatelské požadavky viz. Obr. 2.2. Tyto jsou vždy obsluhovány komponentou View.



Obr. 2.2: Uživatelská interakce

Zde mohou nastav dvě situace, jak se MVC vypořádá s požadavkem zvenčí. Každá tato situace nastává u jiného typu systému. Neodborně se rozdělují na „widgetové“ a „newidgetové“ systémy, což vlastně znamená komponentní a nekomponentní (založené na požadavku) systémy pro tvorbu aplikací.

- Komponentní – Jedná se například o frameworky Silverlight, ASP. NET z rodiny používající jazyk C# anebo Tapestry či JSF, využívající programovací jazyk Java. Zde jsou komponenty schopné ošetřit uživatelský vstup samy ve View. Není však vyjímkou, že tyto systémy využívají Controlleru k zachytávání událostí, které namapuje na funkcionalitu v objektech implementujících návrhové vzory, např. behaviorální vzor Command[8].
- Založené na požadavku (Request based) – Zde patří frameworky jako Spring MVC (jazyk Java) nebo Rails (jazyk Ruby). Zde se o zpracování uživatelského požadavku stará Controller, jak je u architektury MVC definováno.

Různá funkcionalita Controlleru mezi těmito systémy rozděluje MVC architekturu na její odnože: MVC, MVP (Model – View – Presenter).

2.3.3 Variace MVC architektury

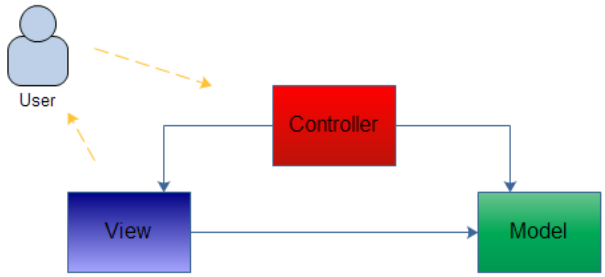
Jelikož je webová aplikace nejrozšířenější strukturou s vzorem s odděleným vstupem a výstupem, MVC se aplikuje v těchto systémech. Vzhledem k faktu, že většina těchto aplikací je tvořena komponentními frameworky, používá se především vzor MVP. Nyní náhled na webové použití této architektury.

2.3.3.1 Vzor MVC

Ve webovém prostředí se chovají komponenty trochu jinak, než je tomu v prostředí desktopových aplikací.

- Model se nemění, obsahuje aplikační doménu a také její logiku.
- View se zde stará o vygenerování (X)HTML kódu na serveru, který je poslán a zobrazen na webovém prohlížeči uživatele.
- Controller zde zastává dvě různé činnosti. První akcí je zachytávání a zpracování uživatelského požadavku přes HTTP. Poté jsou instrukce poslány dalšímu Controlleru, který se již stará o aktualizaci Modelu, který spojí s konkrétním View, kvůli zobrazení nových dat[8].

Definování vztahů mezi jednotlivými částmi MVC. Obr. 2.3.



Obr. 2.3: MVC ve webové aplikaci

Popis sledu událostí je následující:

1. Uživatel spouští událost, která je následně zachycena Controllerem.

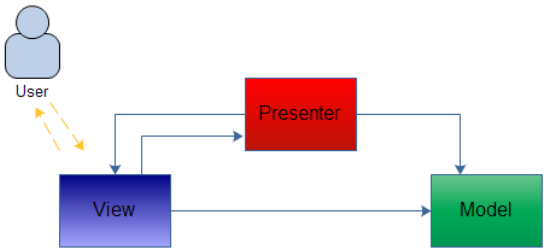
2. Podle předefinovaných návrhových struktur v systému Controller rozhodne, jak bude na danou událost reagovat, jestli zasáhne do Modelu kvůli změně dat anebo nějak ovlivní View pomocí pevné vazby, viditelné na Obr. 2.3.
3. Po ukončení procesů si View zažádá z Modelu změny a ty zobrazí uživateli.

Tento cyklus událostí a akcí se opakuje při každém uživatelském požadavku.

2.3.3.2 Vzory MVP

Jak již bylo zmíněno, vzory MVP se vyskytují u webových aplikací, tvořených komponentními technologiemi(„widgetovými“). Rozdíly mezi MVP a MVC jsou vcelku zásadní.

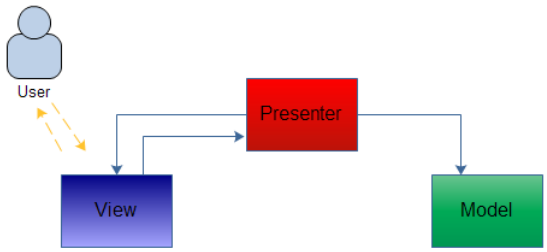
- Model je naprosto stejný jako u vzoru MVC.
- Presenter zde zahrnuje jak prezenční, tak aplikační logiku. Může ovládat View pomocí upozorňovacích zpráv předaných Modelu nebo přímou cestou. A také přímo ovlivňuje Model.
- View na rozdíl od MVC zcela kontroluje uživatelský vstup. Pokud uživatel ve View vyvolá nějakou akci, View pouze spraví Presenter, který se stará o její vykonání.



Obr. 2.4: Vzor MVP - Mode - View - Presenter

Jak je patrné z obrázku Obr. 2.4, View zde plně ošetřuje uživatelský vstup, Presenter zase obsluhuje aplikační a prezenční logiku aplikace. U vzoru MVP se můžeme dále bavit o roli View v aplikaci. Není totiž uplně specifická.

- *Vzor Supervisiong Presenter.* V tomto případě byla přidána View zodpovědnost k zobrazování změněných dat v modelu a to nepřímou vazbou mezi nimi. Právě toto upozorňování ze strany modelu je vyřešeno implementací behaviorálního vzoru Observer(Pozorovatel) nebo data bindingem(datovým svazováním, čehož využívá například framework Silverlight).
- *Vzor Passive View.* Zde se View nesvěřuje žádná zodpovědnost, pouze zobrazování dat. Tento vzor postrádá vazbu mezi Modelem a View, Obr. 2.5. Tento nedostatek je nahrazen přenesením funkcionality synchronizace Modelu a View na Presenter. Ten kromě aplikační a prezentační logiky tedy ještě ošetřuje přenos dat.

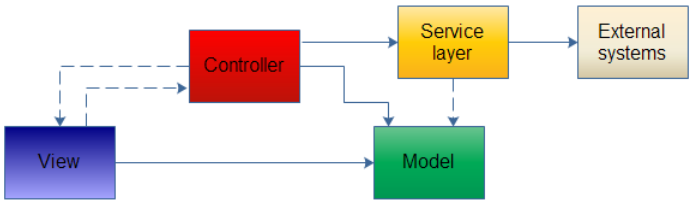


Obr. 2.5: Vzor Passive view

Do této kategorie podle mého názoru patří framework Tapestry. Je to systém komponentní, tedy MVP a jeho vnitřní stavba vyhodnocování požadavků a předávání procesů svědčí o této atchitektuře.

2.3.4 Služby vzoru MVC

Prezentační vzor MVC zajišťuje zobrazování a práci s daty, ale ty musí aplikace někde získat. Taková data jsou uložena většinou na databázovém serveru. Ke komunikaci a získání těchto dat slouží servisní vrstva, jejíž funkcionalita toto zajišťuje. Samozřejmě nemá nic společného se zobrazováním dat, Obr. 2.6.



Obr. 2.6: MVC plus servisní vrstva

Takto již vypadá skutečná architektura webové aplikace. Service layer je servisní vrstva a External systems znamená externí uložiště. Pracuje s reálnými daty a provádí všechny běžné úkony: načtení, změnu, smazání a vytvoření záznamu[8].

2.4 Framework

Framework (pracovní rámec) je softwarový nástroj slučující dohromady návrhové vzory a objektově – orientované programování. Je to znovupoužitelný programový návrh vyjádřený množinou abstraktních tříd a vztahů mezi jejich instancemi[9]. Vývojář může framework přizpůsobit určité aplikaci použitím a složením instancí tříd frameworku.

Framework určuje architekturu aplikace. Definuje její strukturu, rozčlenění do tříd a objektů, klíčové vlastnosti mezi nimi a jejich kontrolu. Díky těmto předdefinovaným návrhovým parametrům se může vývojář soustředit na specifické rysy aplikace. Framework se zabývá obecně známými návrhovými rozhodnutími patřícími do domény rysů určitě aplikace. Je zde také kladen důraz na opětovné použití naimplementovaného návrhu řešení problému.

Abychom rozlišili mezi klasickou programovací knihovnou a frameworkem, musíme

vědět, jak framework využívá některé specifické vlastnosti, které knihovny nemají.

- *Inverze kontroly* – framework zde obstarává celkový aplikační kontrolní tok, který uživatele neovlivní. Jedná se o předdefinovaný scénář rolí a zodpovědností mezi aplikačními prvky[9].
- *Rozšířitelnost a znovupoužitelnost*. Každý framework umožňuje vyprodukovat vlastní třídy a metody buďto napsáním vlastních, anebo rozšířením implicitních. Tato vlastnost se nazývá rozšířitelnost. Takovéto kódy lze uložit a kdykoliv použít znova, zde je to vlastnost zvaná znovupoužitelnost. Na druhou stranu však nelze přepsat kód frameworku, lze jej pouze rozšířit.

2.4.1 Principy Frameworku

Frameworky jsou složeny ze dvou částí: *frozen spots* and *hot spots*. Jelikož doslovné přeložení je velice nejasné, ponechám jejich interpretaci v angličtině.

- *Frozen spots* definují celkovou architekturu frameworku. Ta je složena z implicitních komponent a funkcionality, která definuje jejich vztahy a vazby. *Frozen spots* jsou neměnné části frameworku a zůstávají stejné i při jejich použití v aplikaci.
- *Hot spots* jsou ty části frameworku, které designer vytvořil při tvorbě projektu nebo k přizpůsobení potřeb aplikace. Můžou to také být komponenty či funkcionalita z jiného frameworku, který je použit ve spolupráci s dalšími frameworky. *Hot spots* jsou implementovány pomocí vlastností jako polymorfismus, či delegování.[9] Různorodá aplikační specifikaci vyžaduje většinou zapojení jinak zaměřených frameworků, jeden se stará o prezenční logiku a další například o přístup do databáze.

Návrhové vzory jsou logickým postupem při řešení různorodé problematiky. Framework je poté sada implementovaných logických kroků k řešení specifické tématiky, tzn. realizace návrhových vzorů implementovaná v nějakém programovacím jazyce. I přesto jsou návrhové vzory a frameworky naprosto odlišné pojmy. Návrhové vzory jsou stále logickým postupem a frameworky zase fyzickou realizací několika návrhových řešení v oblasti počítačové vědy[9].

2.4.2 Webové aplikační frameworky

Webový aplikční framework je softwarový nástroj, který pomáhá vývojářům při budování webové aplikace. Tyto frameworky poskytují základní funkcionalitu běžnou pro většinu webů, jako uživatelský management, přístup k datové vrstvě a systém vytváření šablon. Při užití správného frameworku ušetří vývojář řadu času při návrhu webové aplikace.

2.4.2.1 Vlastnosti webových frameworků

Webové aplikační frameworky se mohou skladbou vlasností lišit v závislosti na tom, pro jaký druh navrhované aplikace jsou použity. Jejich základní předdefinovaná funkcionalita je více či méně stejná.

- *Datová persistence* – Každá webová aplikace je v mnoha případech tvořena kromě několika statických také sadou dynamicky generovaných stránek. Tyto stránky získávají data z modelu aplikce, který je obrazem nějaké datové struktury existující buď na lokálním

nebo databázovém serveru.

- *Uživatelský management, session a bezpečnost* - Složitější webové aplikace nabízejí možnosti přihlašování a ukládání uživatelských nastavení. Takže tato vlastnost zajišťuje uživatelovo rozpoznání, poté přiřazení jeho práv a přístupů k daným stránkám a nakonec tzv. session, která udržuje uživatelův stav po dobu aktivní práce s aplikací nebo dokud není odhlášen. Tato schopnost se využívá ve všech systémech, kde existuje možnost přihlášení. Obecně jsou to aplikace, které skladují citlivější údaje nebo ošetřují omezený přistup. Autorizace se týká také bezpečnosti. Někteří uživatelé mohou navštívit a pracovat pouze s těmi stránkami, které jim systém autorizací povolí. Zde patří například autorizace podle rolí uživatele.
- *Cache (sklad)* - Tato vlastnost slouží k zvýšení výkonu, ukládáním nedávno zobrazených stránek do zásobníku, ze kterého můžou být poté zobrazeny bez nutnosti jejich kompletního znovu vytvoření. Vždy tyto *cache* drží instance stránek po omezenou dobu.
- *Vytváření šablon* – při tvorbě dynamických stránek se používá čistý (X)HTML jazyk a k němu je přidána část generovaná z kódu, který obsahuje šablona. Toto generování je založeno na přítomnosti proměnných v šabloně. Dovoluje snižovat počet stran potřebných k zobrazení různých dat na webu.
- *Mapování URL* – Z pohledu moderních aplikací je velmi zásadní použití SEO, česky optimalizace vyhledávače což znamená, že pokud uživatel zadá nějaký výraz do vyhledávače, hledání se zaměří na dokumenty, které obsahují správně nastavené výrazy pro vyhledávání. Jinými slovy, tato vlastnost přemapuje URL v aplikaci pomocí předdefinovaného regulérního výrazu a ve vyhledvači se URL zobrazí jako mnohem přehlednější a vyhledávanější.

2.4.2.2 Architektura webových frameworků

Existuje hodně architektur frameworků, v této práci se zaměřuji na ty s architekturou MVC, tzv. *pull* and *push* frameworky podle toho jak zpracovávají uživatelský požadavek.

- *„Push – based“* nebo také „*Request – based*“ (založené na požadavku) – Z pohledu MVC architektury je View zaregistrován u Modelu a při změně dat Model posílá nové informace do View. Zde si View nezažádává nová data z Modelu, ale sledem procesů mu jsou poslána z Modelu. Zástupci frameworků jsou Struts, Ruby on Rails, Spring MVC, Stripes, ...
- *„Pull – based“* nebo také „*Component – based*“ (komponentově založené) - Zde se View stará o aktualizaci dat sám, musí sám posílat žádost do Modelu. Na úrovni webové aplikace se jedná o zaslání HTTP požadavku z View do Controlleru či Modelu na základě uživatelovy interakce. Model poté odpoví zasláním aktualizovaných dat. Zde figurují frameworky jako JSF, Tapestry, ASP. NET, ...
- *Rich Internet Application* (Bohaté internetové aplikace) – Se soustředí především na použití vizuálních a grafických prvků v aplikaci. Jejich zástupci jsou např. SilverLight, JavaFX, Flex nebo OpenLaszlo.
- *One size fits all* (jeden vyhovující všem) – Frameworky splňující kritéria více druhů aplikací. Záastupcem je Grails.

2.5 Shrnutí

Vstoupili jsme do problematiky webových aplikací a nastínili hlavní rysy těchto technologií. Jak frameworky pracují s návrhovými vzory, jakým způsobem jsou postravené jejich architektury, které umožňují tvorbu přehlednějších struktur aplikací a také usnadňují jejich správu. Jelikož se své práci zkoumám především framework Tapestry a jeho soupeře, postupně budu uvádět jednotlivé specifika jejich funckionality, struktury a architektury.

3. Požadavky na moderní webové aplikace

Požadavky na moderní webové aplikace bych rozdělil do dvou sfér. V první sféře se nacházejí aplikační logika, funkcionalita, struktura a doména aplikace. Ve druhé pak její podoba, využití a spolehlivost a zpřístupnění. Abych tento pohled více přiblížil rozdělíme si ho na dvě nezávislé entity. První je pohled vývojáře a druhý pohled uživatele. Všechny tyto správné vlastnosti by plně funkční a přehledná webová aplikace měla mít a starostí jejího designéra (či týmu) je se postarat o jejich implementaci. Pomiňme fakt, že čím přesnější jsou zákaznické požadavky na aplikaci, tím přehlednější a spolehlivější je realizace návrhu.

3.1 Vnitřní funkční požadavky aplikace

Z prvního pohledu je patrné, že aplikace musí splňovat požadavky týkající se především funkcionality. Některé z nich již byly nastíněny v předchozí kapitole, ale je jich podstatně více. Lze říct, že o všechny tyto podstatné děje a stavy uvnitř aplikace se již postaral někdo jiný a své znalosti abstraktizoval ve frameworku, ve kterém je aplikace utvářena. Tím pádem se o řešení plnění nízko úrovňových instrukcí, implementací návrhových vzorů apod. nemusí vývojář starat a soustředí se na implementaci funkčních požadavků.

3.1.1 Architekruta aplikace

Prvním krokem k dobře fungující aplikaci je její základ, tedy architektura. Při její volbě na samém počátku návrhu musíme vědět k čemu bude aplikace určena. Jedná-li se o primitivní aplikaci poskytující pouze statické stránky, pak není nutné používat složitých architektur.

Pokud se jedná o aplikaci, která bude komplexnější, třeba e-shop nebo informační portál, je vhodné volit již specifickou architekturu, která pokryje všechny nároky jak na datovou, tak aplikační a nakonec i prezenční složku. To znamená rozčlenění aplikace na vrstvy (datové, aplikační a prezentační), definování vztahů mezi nimi,popis procesů v systému až po jednotlivé aplikační komponenty[7].

3.1.2 Členitost

Členitost nebo modularita je využita především ve frameworcích při návrhu aplikace. Dovoluje rozdělit aplikaci do ustálených logických částí tak, že tyto části či moduly, mohou být použity kdykoliv znova. Další její výhodou je rozšířitelnost podle uživatelských potřeb. Jedná se vlastně o tzv. *hot spots.* Dále sebou tato funkcionalita přináší možnost rozdělit práci při implementaci aplikace do rolí. Takže se každý člen týmu může starat o jiný problém, aniž by se navzájem ovlivňovali. Celkově se díky členitosti aplikace zvýší její přehlednost.

3.1.3 Uživatelská session

Komunikace mezi klientem a serverem probíhá přes bezstavový protokol HTTP. Jedná se o odpovědi serveru na klientovy požadavky, aniž by věděl jestli se jedná o stále toho jednoho uživatele. Na úrovni aplikace existuje tzv. s*ession,* kterou využívá většina aplikací. Slouží k rozpoznávání daného uživatele na základě serverem přiděleného indentifikátoru. Uživatel nemá vlastně o této session ani ponětí, ale jeho prohlížeč ano. Ten si ji uloží ve formě tzv. *cookies* a

odesílá ji serveru s každou novou žádostí. Takto server pozná o koho se jedná a také se podle toho zachová, zde jsou pak realizovány případné autentizace jako bezpečnostní opatření.

3.1.4 Uživatelský management

Nástroje pro autentizaci a správnou autorizaci patří do každé komplexnější aplikace. Při autentizaci je uživatelem vyslán požadavek s přiloženými údaji jako uživatelské jméno a heslo přes nejlépe zašifrovaný kanál (HTTPS) na server a ten se snaží rozpoznat podle zaslaných údajů s přihlédnutím do databáze s uloženými informacemi o uživatelích, zda-li je to právě ten uživatel za kterého se vydává. Pokud ano, server zasílá zpátky session, kterou se přihlášený uživatel prokazuje.

Při každé takové autentizaci je uživatel zařazen do určité skupiny podle jeho autorizace. Autorizace určuje rozsah možností, které může uživatel v systému provést, které stránky si může prohlížet, jaké operace provádět apod. Platí pravidlo, že by uživatel měl mít přístup pouze k těm funkcím, které potřebuje.

3.1.5 Datová vrstva

Žádná aplikace by nemohla fungovat, pokud nemá přístup ke skutečným datům, uložených někde na databázovém serveru nebo v jiném systému.

Nejjednodušším přístupem do databáze je přes API systému řízení báze dat (ŠŘBD).

Další možností je přístup do databáze přes programové knihovny, které již poskytují komplexnější rozhraní pro vstup a práci s daty díky naimplementovaným funkcím. Jsou však naprosto závislé na použitém ŠŘBD a při jeho změně je třeba přepsat všechny již implementované metody[7].

Nejvyužívanější přístupovou metodou k datům jsou ovladače ODBC anebo v jazyce Java pak JDBC. Jedná se o specializované knihovny s funkcemi pro přístup do databáze. Jsou častým základem frameworků, které využívají jejich metody přístupu a dodávají další, s vlastní funkcionalitu.

3.1.6 Validace vstupních parametrů

Důvodem validace formulářů webové aplikace jsou velmi oblíbené kybernetické útoky, kterými se mohou útočníci dostat k citlivým informacím, uloženým v databázi (SQL injection) anebo třeba do stránky zavést škodlivý scriptovací kód (XSS – cross site scripting). Pomocí validace lze v těchto místech zavést protiopatření, která tento škodlivý kód přefiltrují a aplikaci tak ochrání. Validace je mechanismus ošetřující vstupní řetezce podle regulérních výrazů a povolující pouze ty, které splňují podmínky. Takže například email řetězec musí obsahovat nějaký počet znaků + @ + nějaký počet znaků + . + dva až tři znaky.

3.1.7 Bezpečnost

Bezpečnost aplikace by nejlépe měla být zajištěna jak po hardwarové stránce, tak i té softwarové a poté i svou politikou. Autentizace, autorizace a validace zajišťují bezpečnostní politku proti neoprávněnému vstupu do aplikace. Správné uložení šablon, stránek a logiky na zajištěný webový server zase zajistí nedostupnost služeb bez dostatečného oprávnění. Dobře zabezpečené aplikace

používají jednoduchou bezpečnost, složitější procesy kontrolování totiž mohou zavinit neochotu uživatelů k navštěvování takovýchto stránek. Žádná aplikace není nikdy sto procentně zabezpečená a je vždy tak bezpečná, jako je její nejslabší komponenta.

3.2 Vnější funkční požadavky

Každá aplikace je určena specifickému návštěvníkovi, proto se vyplatí zaměřit své úsilí na zviditelnění aplikace na webu. Také nesmíme opomenout pohodlné ovládání a multijazyčnost, která zpřístupňuje web také v cizím jazyce, s čímž narůstá počet uživatelů nebo potencionálních zákazníků.

3.2.1 SEO

Česky optimalizace pro webové vyhledávače. Je to taktika pro zviditelnění obsahu aplikace mezi konkurencí při vyhledávání přes webové prohlížeče. Tato metoda je poměrně nová, ale významná z hlediska dostupnosti a ceny. Většinu faktorů ovlivňujících pozici stránek zobrazených při hledání klíčových výrazů samo chování aplikace neovlivní. Je to věcí designerů a odborníků na SEO, jako příklad bych uvedl: Validní (X)HTML kód, nepoužití Javascriptu u navigačních položek, flashové animace a rámce. Všechny tyto „vady“ neumožňují indexaci pojmů na stránkách a dochází k znehodnocení kvality hodnocení vůči klíčovým slovům a tím pádem degradaci obsahu aplikace v pozici vyhledávání. To co sama aplikace ovlivní je mapování URL adresy, na které SEO klade nemalý důraz. Dlouhé řetezce paramertů za doménovým názvem aplikace SEO hned zahodí, na rozdíl od formátované URL.

3.2.2 Multijazyčnost

Pokud zavedeme multijazyčnost do naší aplikace, zpřístupníme ji dalším uživatelům z rozličných kultůr, zemí. Tato vlastnost je označována jako i18n (z anglického internationalization, kde mezi prvním „i“ a posledním „n“ je právě 18 písmen). Tento poměrně nový požadavek specifikuje země, pro které bude aplikace dostupná. Logicky tak vyplývá, že s čím více jazyky budeme počítat, tím využitelnější aplikace bude. K internacionalizaci patří také lokalizace (l10n – localization), která zajistí zjištění jazykového nastavení uživatele, který aplikaci používá z jeho prohlížeče.

3.2.3 Ajax

AJAX je soubor technologií, které umožňují interaktivní práci s daty na webovým prohlížečem zobrazené stránce bez nutnosti znovunačtení. Některé starší prohlížeče mohou mít s AJAXem potíže, pokud např. nepovolují JavaScript.

AJAX je systém využívající několik technologií, aby dosáhl své interaktivní vlastnosti. První je technologie (X)HTML a CSS (kaskádové styly) pro vytvoření dokumentu a jeho stylistiku, pak JavaScript a DOM pro dynamické změny prezentovaných informací a XMLHttpRequest, která zajišťuje asynchronní výměnu dat s webovým serverem[2].

Výčet požadavků na moderní aplikace není úplný. Každý zákazník může mít jiné představy, ale i přesto tento seznam zachycuje hlavní technické požadavky aplikací dneška.

4. Srovnání Java Frameworků

Požadavky na moderní webové aplikační frameworky se již neskládají pouze z jejich „základních“ řešení funkčních požadavků, ale musejí také zastávat implementaci vnějších požadavků na aplikaci, jak bylo zmíněno v kapitole 3. 2. Takovéto frameworky nabízejí možnosti implemetace uživatelsky přívětivějších podob, atraktivní vícejazyčnosti a uživatelsky „přátelského“ rozhraní.

4.1 Java technologie

Webových frameworků postavených na jazyce Java existuje mnoho. Všechny však mají společné základy postavené na platformě Java enterprise edition. Jejich dalším společným parametrem je komunikace mezi klientem a serverem, založená na stejných mechanismech a využívá servletů a JSP[12].

- *Java EE* – (jinak označována jako J2EE) je platforma pro serverové programování v jazyku Java. Využívaná je ke konstrukci a provozu podnikový a informačních systémů webových aplikací [2].
- *Servlet container* – Je kus kódu, který spolupracuje s webovým serverem. Požadavky přicházející z internetu na webový server jsou tříděny a pokud přicházejí z Java aplikace jsou předány do servlet kontejneru. Kontejner zpracovává tyty nízko úrovňové instrukce týkající se síťování nebo ošetřování paměti využité v aplikací. Díky tomuto postupu se vývojáři mohou zaměřit na psaní servletu pouze pro určité aplikace. V těchto případech webové aplikační frameworky nabízejí již ošetřené zpracovávání nízkoúrovňových instrukcí v jejich API.
- *Java Servlet* – je standardizovaný způsob psaní programovacího kódu v Javě, který může být použit v servletovém kontejneru a těžit ze služeb, které poskytuje. Stejně jako JSP je servlet nezávislý na platformě nebo serveru. Servlety mají také přístup k Java API, kde mají přístup k JDBC nebo ke knihovně specifikých HTTP-tříd.
- *JavaServer Pages* – Nabízí snadnou a rychlost cestu, jak vytvářet dynamickou webovou šablonu. Tato technologie dovoluje rychlý vývoj webových aplikací, které nejsou závislé na serveru ani na platformě. Pomáhá oddělit uživatelské rozhraní se zbytkem aplikace, a tak může designér tyto částí změnit bez ovlivnění jiné.

Podrobné srování frameworků by zabralo na rozsáhlou práci, a jelikož se těmito tématy zabývali jiní, pouze bych uvedl popis každého frameworku, ale předtím uvedu jejich základní strukturu a funkcionalitu. Důvod, proč jsem volil právě tyto frameworky je ten, že jsou nejpoužívanější v oblasti webových aplikačních frameworků využívajících Java technologie.

4.2 Přehled

V tabulce 4. 1[1]. je přehled srovnávaných frameworků.

- *Projekt* – Zde je uveden zaštítovací projekt, který ručí za vývoj a produkci daného frameworku.

1 http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks

- *Současná stabilní verze* – Určuje poslední verzi proejktu, která je spolehlivě testována a funkční.

Projekt	Současná stabilní verze	Licence
Apache Struts	2.1.8.1	Apache 2.0
Apache Wicket	1.4.5	Apache 2.0
JavaServer Faces	2	
Spring Source	3.0.0	Apache 2.0
Stripes	1.5.3	Apache 2.0
Tapestry	4.1.6/5.1.0.5	Apache 2.0

Tabulka 4.1: Projekt, verze a licence

- *Licence* - Týká se možností distribuce, užití a šíření. Většina spadá pod Apache Licence, Version 2. 0[1].

4.3 Základní rysy

V těchto tabulkách 4. 2. a 4. 3[2] se nacházejí základní parametry webových frameworků, samozřejmě těchto kritérií je podstatně více, ale jak jsem již zmínil, každý se s určitým aplikačním požadavekem vypořádává jiným způsobem, a tak je jednoznačné hodnocení vždy subjektivní.

- *Jazyk* – Udává použitý programovací jazyk.
- *Ajax* – Pokud framework umožňuje použití asynchornního JavaScriptu kvůli interaktivnější a atraktivnější práci s aplikací na uživatelské straně.
- *MVC* framework – Uvádí, jestli je daný framework postaven na MVC architektuře.
- *MVC Pull/Push* – Tento parametr vypovídá o frameworku, jestli se jedná o komponentní webový aplikační framework anebo nekomponentní (řízený požadavky).
- *i18n & i10n* – Určuje jaký typ multijazyčnosti a lokalizace daný framework poskytuje. Jedná se o získání lokálního jazyka z HTTP hlavičky a podle aplikace pak možnost přepínání mezi jednotlivými jazykovými variacemi.

Projekt	Jazyk	Ajax	MVC Framework	MVC Push/Pull	i18n & l10n?
Apache Struts	Java	Ano	Ano	Push & Pull	Ano
Apache Wicket	Java	Ano	Modulární, řízený požadavky	Pull	Ano
JavaServer Faces	Java	Ano	Ano, řízený požadavky	Pull	Ano
Spring	Java	Ne	Ano	Záleží na použití	Ano
Stripes	Java	Ano	Ano	Push	Ano
Tapestry	Java	Ano	Ano	Pull	Ano

Tabulka 4.2: Základní parametry

1 http://www.gnu.org/licenses/license-list.html#apache2
2 http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks

O těchto vlastnostech jsem se již zmínil v předchozí kapitole a je velice žádoucí, aby je webové aplikační frameworky poskytovaly, jelikož jsou stavebními prvky každé moderní aplikace.

- *ORM* – Objektově – relační mapování. Mapuje relační entity na objekty v modelu aplikace a umožňuje tak práci s nima v objektově – orientovaném programování a užití těchto tříd v aplikaci jako Model.
- *Zabezpečení* – Uložení dat, stránek a komponent na serveru je více než žádoucí. Metody autentizace a autorizace zase umožňují omezení zobrazování citlivých dat nežádoucím osobám, pokud pro to nemají oprávnění.
- *Užití šablon (Templates)* – Poskytování šablon je definice toho jak se má View zobrazit v uživatelském prostředí. Tyto šablony mohou obsahovat komponenty (předdefinovaný kód s určitou funkcionalitou a formou tvořenou HTML kódem a styly) a po požadavku ze strany uživatele být doplněny patřičnými proměnnými, a poté renderovány jako HTML kód na stranu klienta, kde se zobrazí podle daného požadavku. Velice ulehčují práci designérům.
- *Formulářová validace* – Slouží k omezení škodlivého kódu vloženého do aplikace přes možné vstupy jako formuláře, knihy návštěv atd. a ke kontrole zadaných uživatelských údajů.

Projekt	ORM	Zabezpečení	Užití šablon (Templates)	Formulářová validace
Apache Struts	Ano	Pomocí JEE kontejneru	Ano	Ano
Apache Wicket	Ano, pomocí rozšíření	Ano	Ano	Ano
JavaServer Faces	integrace s Hibernate	Ano	Ano, Facelets	Ano, ale konfigurace
Spring	Hibernate	Spring Security	Tiles, Velocity	Běžné Validace
Stripes	JPA, Hibernate	Ano s rozšířením	Ano	Ano
Tapestry	integrace s Hibernate	tapestry5-acegi library	Ano	Zabudované validace

Tabulka 4.3: Základní parametry II

Po shrnutí, tyto všechny možnosti by měl moderní webový framework nabídnout. Vybrat si ten nelepší nejde, jelikož není jasné, který je neužitečnější, závisí čistě na zkušenostech každého vývojáře, na požadavcích aplikace a prostředcích firem.

4.4 Popis webových aplikačních frameworků

V následujícím rozboru je stručná charakteristika daných frameworků. Rád bych tímto, pouze přiblížil jejich vlastnosti a specifika.

4.4.1 Struts 2

Původně byl tento framework znám jako WebWork 2, ale později se komunity Struts a WebWorku rozhodly spojit a vznikl Struts 2.

Struts2 je postaven na strategii, která zahrnuje celý aplikační vývoj: návrh, implementaci a správu. Je snadno rozšiřitelný, protože každá třída obsahuje rozhraní, které nenutí programátorovi psát celé třídy. Navíc základním třídám je dodáno extra vlastností. Samozřejmě poskytuje možnost

rozšíření o další vlastní třídy. Požadavky frameworku jsou: Servlet API 2.4, JSP API 2.0 a Java 5.

Vlastnosti:

- POJO objekty
- spolupráce se Spring MVC frameworkem
- podpora JSP, JSF, JSTL, …
- na úrovni modelu spolupracuje s JDBC, EJB, Hibernate

4.4.2 Spring

První verze byla napsána Rodem Johnsonem. Vydána byla pod Apache 2.0 licencí 2003. Současná verze nese číslo 3. 0. 2. Základní funkce Springu mohou být použity jakoukoliv Java aplikací, její rozšíření vedou k vývoji webový aplikací na vrcholu platformy Java EE. Ačkoliv Spring neimplementuje žádný specifický programový model, stal se velmi populární součástí jiných frameworků používaných v produkci. Může například nahradit nebo doplnit EJB model[1].

Vlastnosti:[1]

- Aplikace by neměla být závislá na Spring API.
- Integruje jiná prostředí a nástroje pro komplexní řešení: Hibernate pro ORM řešení, Toplink, JDO.
- Obsahuje neinvazivní kontejner nebo-li řešení IoC pro aplikační objekty
- Zabudovaný MVC frameworkem
- Vrstva pro správu transakcí
- Využitelný pro skoro každou aplikaci
- Nevýhodou je například nutnost nalinkování knihoven pro podporu AJAXu

4.4.3 JavaServer Faces

JavaServer Faces je velmi oblíbený webový aplikační framework s aktuální verzí 2. 0 vydanou 28. 6. 2009 pracující na Java EE 6.

JSF jsou založené na architektuře MVC řízené požadavky s designem založeným na komponentách. Jako šablony pro zobrazení jsou použity XML soubory alias Facelety samozřejmě založené na JSP. FacesServlet zase spojuje požadované stránky s logikou, konstruuje komponenty, zpracovává události a odpovídá na požadavky klientovi. Podporuje stavovost na klientské nebo serverové straně. Označuje se vysokou abstraktizací na straně templatů u View.

- Šablonově - komponentní systém pro rychlé vytváření komponnent bez Java tříd.
- Několik možností jak implementovat AJAX v JSF (umístění v šabloně, komponentách).
- Integruje sjednocený výrazový jazyk (EL), který je základem pro JSF funkce.
- Serverový událostní model pro přidělování posluchačů k události.
- Možnost výběru ze dvou XML tagových knihoven při zobrazení JSF View. Buď Facelety anebo JSP.

- Stavový management podporující požadavky, session, flash a Java Bean.
- Nesnadná tvorba nativních komponent a nelze kombinovat komponentní sady
- Při použití v jednoduchých případech aplikace zbytečně komplexní řešení

4.4.4 Stripes

Stripes je prezentační framework pro stavbu webových aplikací, který využívá poslední Java technologie a je postaven na MVC vzoru. Zaměřuje se na ulehčení vývoje aplikací v Javě oproti Struts tím, že využívá anotace a generické datový objekty. Celkově je zaměřen na odpoutání se od náročných konfigurací v aplikaci. Opět je zde možnost rozšíření k dalším frameworkům.

Vlastnosti:

- Anotace, díky kterým nemusí programátor přiřazovat stránky k akcím.
- Využívá data binding.
- Možnost lokalizace pouze s JSP odkazy – snadná internacionalizace.
- POJO, Ajax a JPA pro práci s modelovou vrstvou.
- Maskování URL.

4.4.5 Wicket

Wicket spadá opět pod Apache projekt. Je to komponentně založený webový aplikační framework podobný JSF a Tapestry. Napsán byl Jonathanem Lockem v roce 2004.

Využívá tradičních vzorů jako MVC i komunikace mezi jednotlivými komponenty MVC. Snaží se využívat POJO a HTML pro tvorbu komponent, aby se vyhnul použití složitějšího XML[1].

Vlastnosti:

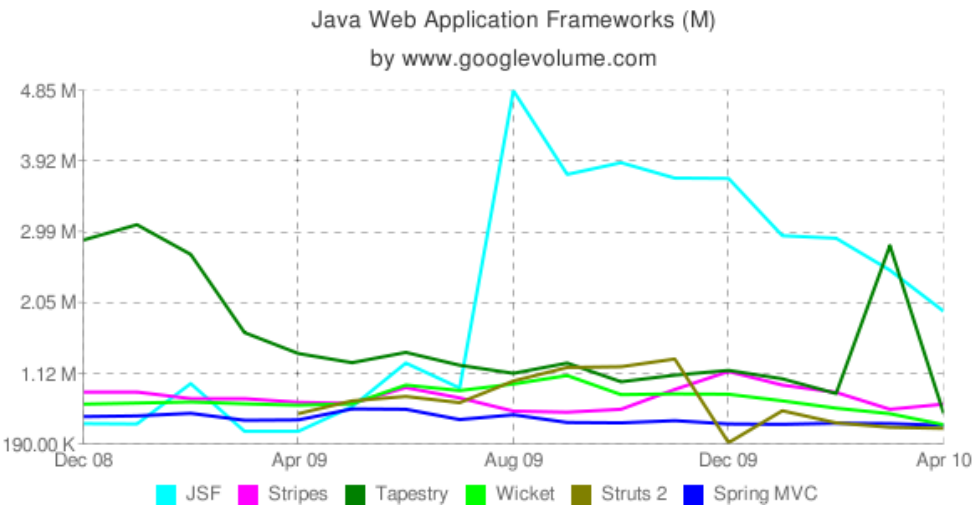
- Bezpečnost – Defaultní nastavení bezpečnosti. URL nevystavuje citlivé informace ani komponentní cesty. Snaží se kriptovat URL a skrývat informace kolující mezi sessiony.
- Jednoduchá lokalizace a validace formulářů.
- Šetření paměťovými a síťovými prostředky. Inicializuje pameť při použití nějaké části modelu a po skončení práce ji opět uvolní.
- Lokalice – Stránky, obrázky a zdroje mohou být lokalizovány.

4.5 Srovnání

Jak jsem již naznačil v úvodu do této kapitoly je opravdu těžké říct, který framework je lepší. Spíše bychom se měli zamyslet nad tím, jak uchopit tyto nástroje pro tvorbu webových aplikací a využít jejich vlastností při řešení zadaných problémů – implementaci aplikací. Z čistě statistických účelů uvedu dva grafy, které zobrazují četnost vyhledávání názvů frameworků v google[1] (Obr. 4. 1) a poté četnost vyhledávání prací spojených s těmito frameworky na indeed[2] serveru. (Obr. 4. 2)

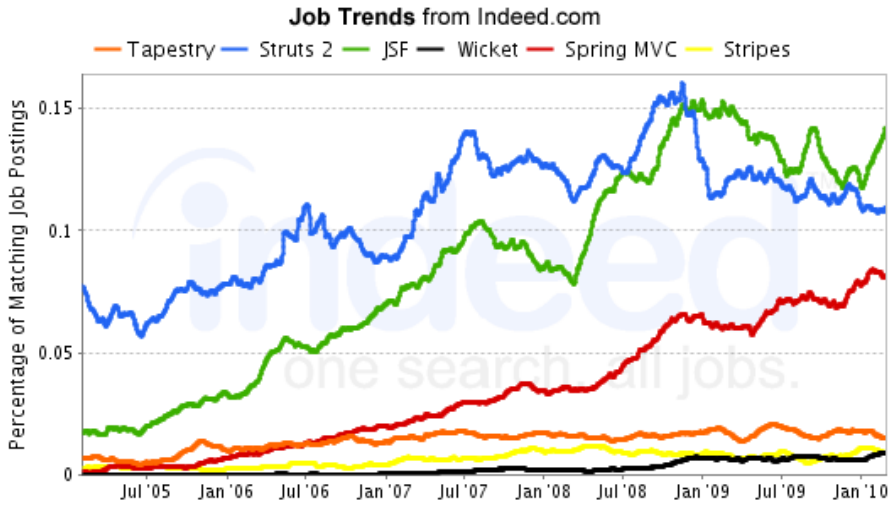
1 www.google.com
2 www.indeed.com

- Četnost vyhledávání na www.google.com, klíčový výraz: „Java Frameworks“.



Obr. 4.1: Graf četnosti vyhledávání klíčových slov na google.com

- Četnost vyhledávání na www.indeed.com ,klíčový výraz: „Tapestry, Struts 2, JSF, Wicket, Spring MVC, Stripes“.



Obr. 4.2: Pracovní trendy spojené s klíčovými výrazy porovnávaných frameworků

Z prvního grafu se dá usoudit nárůst hledání problematiky spojené s JSF a tím pádem jejím větším využitím při vývoji aplikací. Za JSF se poměrně vysoko drží Tapestry, ostatní frameworky mají nižší a relativně stejnou četnost vyhledávání.

U druhého grafu je situace poněkud jiná. Jedná se o použití výrazů daných frameworků v prostředí nabídek pracovních pozic. Zde se výsledky blíží realitě, s kterou jsem se setkal při své práci a to oblíbenost a doporučovanost jinými programátory. Výsledky ukazují poměr využívání frameworků a zde jasně vede JSF a Struts 2 a za nimi Spring. Pak následuje Tapestry a ostatní.

Se všemi výše popsanými webovými požadavky se Tapestry vypořádává po svém. V případě ORM nabízí možnosti integrace Hibernate nebo jiného *backend* frameworku při implementaci mapování. V otázkách bezpečnosti aplikace nabízí integraci s *acegi* knihovnou

security tools, popř. systémem anotací.Tapestry používá vlastní *template engine* s XML prvky. V případě validací nabízí použití validátorů v každé komponentě MVC a také vyhodnocování na klientské nebo serverové straně. Podporuje vlastní implementaci l10n a i18n.

Na druhou stranu chybí vysoká abstraktizace v šablonách, kde ji mnohem více využívá JSF. Také architektura aplikace není příliš flexibilní jako u frameworku Spring MVC. Velmi spekulativní je míra konfigurace nutná pro chod hlavních aplikačních prvků na rozdíl od Stripes, protože může na druhou stranu nabídnout mnohem větší flexibilnost při využívání služeb z cizích prostředí.

Další nevýhodou je složité nastavování importace frameworku do jiných aplikací, hlavně EJB, vytvářených v různých IDE. Struts, JSF a Spring MVC tuto možnost nabízí implicitně bez jakékoliv nutnosti předchozích instalací (především se jedná o IDE NetBeans 6. 8).

5. Tapestry 5 v moderních aplikacích

Tapestry je objektově – orientovaný open source webový aplikační framework s komponentní technologií pro vytváření dynamických, robustních, vysoce členitých webových aplikací v Javě s ohledem na architektonický vzor MVC. Tapestry doplňuje a buduje aplikaci na standardních Java Servlet API, a tak spolupracuje s jakýmkoliv servletovým kontejnerem nebo aplikačním serverem. Tapestry využívá modulární přístup k vývoji aplikací tím, že spojuje komponenty UI webových stránek s příslušnou Java třídu silným vázacím mechanismem – jmennou konvencí.

Tapestry rozděluje aplikaci do množství stránek, kde každá stránka obsahuje komponenty. Ty nabízejí konzistentní strukturu, která umožňuje frameworku ošetřovat klíčové funkce aplikace jako rekontrukce URL a její použití (s ohledem k SEO nebo vlastním nastavením), persistentní uložení stavů na klientské nebo serverové straně, ošetření uživatelských vstupů, lokalizaci, multijazyčnost a chybová hlášení. Vývoj Tapestry aplikací zahrnuje vytváření XML šablon v kombinaci s přiměřeným množstvím Java kódu v šabloně.

Při použití Tapestry vytváříme aplikaci z pohledů objektů, jejich metod a atributů, namísto posílání hodnot atributů přes URL a dotazovací parametry. Není zde ani opomenuta možnost jednoduchého vytváření vlastních komponent.

V Tapestry lze implementovat aplikaci jednoduchou, sestávající z několika stránek stejně dobře jako masivní aplikaci obsahující stovky individuálních stránek, navrženou a implementovanou v týmech. Tapestry dokáže také integrovat jakýkoliv aplikační *backend* (spojující část s datovou vrstvou a databází), včetně JEE, Spring nebo Hibernate. Tapestry je vydána pod Apache Software Licencí 2. 0. Chtěl bych uvést, že v těchto následujících kapitolách čerpám ze zdrojů [12] a [13] pokud není uvedeno jinak.

5.1 Historie

Tapestry framework byl vytvořen Howardem Lewisem Shipem a poté adoptován společností Apache Software Foundation. Tapestry vzniklo v roce 1999. Do širšího povědomí proniklo v roce 2005, kdy vyšla třetí verze. V roce 2006 se objevila verze 4 zpětně nekompatibilní s Tapestry 3. Tato verze byla postavena na HiveMind, což je další projekt H. L. Shipa. Toto spojení dovolilo rozšířit možnosti konfigurace Tapestry a přídávat další funkcionalitu díky HiveMindu, což je vlastně kontejner s kontrolní inverzí. Byla přidána i uživatelská validace.

Po krátkém čase začal H. L. Ship pracovat na úplně nové verzi 5, která jako předchozí není zpětně kompatibilní, ale je postavená na zcela novém sytému podle předchozích zkušeností a do budoucna se již počítá se zpětnou kompatibilitou.

Tapestry 5 byl navžen jako současný framework, který nabízí přátelské uživatelské funkce a snadnou implementaci. Třídy stránek jsou čisté POJO, které nemusí dědit z žádné specifické frameworkové třídy. Byly přidány Java anotace a framework se tím stal přizpůsobitelný uživatelům.

5.2 Zdroje

Velkou nevýhodou Tapestry je dostupnost informací. Jelikož se jedná o nezaběhnutý framework,

nebyl v minulosti příliš nasazen při tvorbě pro velké známé společnosti, a proto postrádá širší dokumentaci a větší komunitu, jakožto i literární zdroje, ikdyž v poslední době se snad věci změní k lepšímu. Největší množství informací se dá nalézt na stránkách projektu tapestry.apache.org [13] , v knize z roku 2007 od Alexandra Kolesnikova [12], knize H. L. Shipa z roku 2004 *Tapestry In Action* (zabývá se Tapestry 4), Enjoying Web Development with Tapestry od Ta Lok Tonga (Tapestry 4.1), nedávno vydané publikaci v němčině a nakonec na wiki stránkách projektu a v ruzných článcích a blozích. Nicméně větší informovanost a dostupnost aktuálních materiálů by projektu Tapestry pomohla k efektivnějšímu nástupu, coby webového nástroje pro vývoj aplikací.

5.3 Využití Tapestry 5 v moderních webových aplikacích

V 3. kapitole jsem nastínil funkční požadavky moderních webových aplikací. V předchozí kapitole jsem se zabýval, jak je implementují jiné Java frameworky a nyní rozeberu, jak se s nimi vypořádá Tapestry 5.

5.3.1 SEO

Od verze 5.1.0.1 Tapestry podporuje přepisování URL. Příchozí požadavek a spojení generované Tapestry může být přepsáno za použití stejného API. Je založeno na řetězení URLRewriterRule rozhraní. Každé z těchto pravidel může být, ale také nemusí, použito při zpracování požadavku. To znamená, že můžeme sami ovlivnit přepisování URL na základně námi vytvořených pravidel a ovlivnit tak indexování aplikace při vyhledávání. Takže direkce URL z hlediska SEO je čistě v našich rukách. Podpora přepisování URL je konfigurována Tapestry IoC skrze spolupráci s URLRewriterRule rozhraním a URLRewriter službou.

5.3.2 Ajax

Tapestry zahrnuje sofistikovanou podporu JavaScriptu a AJAXu založenou na knihovnách JavaScriptu *Prototype* a *Scriptaculous*. Tyto knihovny jsou zabaleny v samotném Tapestry frameworku. Účelem je poskytnutí základních komponent v aplikaci při její tvorbě. Jelikož se tyto knihovny nemusí nijak separovaně stahovat či instalovat není nutná žádná dodatečná konfigurace. AJAX je v Tapestry spouštěn jinými předdefinovanými komponentami jako je *ActionLink*.

5.3.3 MVC framework

Tapestry implementuje architektonický vzor MVC a přímo tak podporuje tvorbu aplikací v této architektuře.

5.3.4 MVC Pull/Push

Tapestry je komponentově založený framework, kde uživatel předává požadavek do View a přes Controller probíhá proces získání a obdržení dat z Modelu, které pak Controller pošle do příslušného View.

5.3.5 i18n & l10n

Tapestry umožňuje podle uživatelského jazykového nastavení vybrat příslušný jazykový katalog a

nastavení, který sestává z textů, hlaviček tabulek, templatů a také obrázků. Lokalizace ošetřuje toto propojení a také rozděluje přehledně prezentované informace a texty od kódu a šablon aplikace. Právě díky jmenné konvenci a katalogům zpráv. Tapestry pak přiřazuje vybrané jazykové nastavení k správným zdrojům.

5.3.6 ORM (Objektově – relační mapování)

Tapestry lze poměrně snadno integrovat s více druhy *backendového* frameworku či platfromy. Nejznámějšími jsou Hibernate, JEE nebo třeba Spring MVC a nabízejí základní možnosti práce s informacemi databáze jako: vytvoření, změnu, smazání a zobrazení.

5.3.7 Zabezpečení

V každé aplikaci je potřeba ošetřit nějakým způsobem možnosti zobrazování a trasování mezi stránkami, kdo se může kam dostat. Budeme – li se držet základních pravidel (uživatelé uložení v databázi, uživatelské role, některé stránky na nedostupných místech, autentizace, autorizace apod.) bezpečnost aplikace bude větší. Tapestry umožňuje všechny tyto základní prostředky, ale také něco navíc. Jedná se o integraci s bezpečnostními prvky jiných frameworků, což samozřejmě není jenom výsadou Tapestry anebo také vytvoření vlastních annotací pro roztřídění možnosti k jejich přístupu a zobrazení.

5.3.8 Užití šablon *(*Templates*)*

Figurují zde soubory komponentních šablon spojené se třídou stránky nebo komponenty, které obsahují značení pro dané komponenty. Šablony jsou dobře formátované (*well - formed*) XML dokumenty používající standardní (X)HTML a čerpající rozšíření k tagům přes Tapestry jmenný prostor. Takové soubory jsou pak uložené ve formátu *.tml* (Tapestry Markup Language).

5.3.9 Formulářová validace

Tento funkční požadavek dnes již každé webové aplikace je využíván při přenosu informací od uživatele do aplikace, ať už při registraci, vyhledávání anebo přihlašování. Tapestry nabízí rozličné množství validací formulářů. Validace je deklarativní, to znamená, že programátor definuje jaká omezení a kontrolu použít k jakým atributům. Tapestry se poté postará o její provedení. Validační logika může být napsána jak ve View, tak i v *backendu* a to přímo ve třídě, kde se aplikuje na určitý atribut a to formou anotace. Při chybovém hlášení je uživateli poslána infromace o špatných hodnotách a Tapestry také označí, které pole obsahuje tyto špatné údaje. (Za pomocí kaskádových stylů.)

5.3.10 Znovupožitelnost komponent

Tapestry stejně jako jiné frameworky nabízí možnost znovupožitelnosti komponent. Nejedná se pouze o rozšiřování tzv. *hot spots* jako v jiných systémech, ale také nabízí možnost vlastního vytvoření komponent. Takto vytvořené části lze uložit do knihoven a poté importovat do dalších projektů bez ohledu na použitou platformu.

5.4 Shrnutí

Tapestry nabízí možnosti řešení všech běžných požadavků webových aplikací, týkající se jak vnitřní, tak i vnější funkcionality, stejně jako jeho největší protivníci JSF nebo Struts. Avšak v implementaci těchto řešení jde vlastní cestou, jindy využívá integrace s jinými systémy. Snaží se omezit některé zaběhnuté a ne příliš oblíbené techniky a problém řeší po svém. Což někdy vede k náročnější implementaci problému. Na druhou stranu nabízí jednodušší řešení než oponenti, hlavně v konfiguraci aplikace a postavení programování na vyšší úroveň abstrakce tzn.že se nezabývá se tolik nízko úrovňovými instukcemi a vývojář se může soustředit na implementaci specifické problematiky.

Kromě toho nabízí velké množství předdefinovaných služeb a také způsoby, jak tyto služby plně využívat nebo přepisovat, což vnáší do celého frameworku nové možnosti.

6. Základní rysy Tapestry5

Aplikace Tapestry je množina interaktivních stránek spravovaných a řízených frameworkem. Každá stránka sestává z šablony, což je XML dokument a stránkové třídy POJO, která nedědí z žádné jiné třídy a neimplementuje žádné rozhraní. Stránky mohou obsahovat tzv. expanze a komponenty. Stavy aplikace a její činost je řízena vlastnostmi tříd a k zpracování a obměně jejich hodnot dochází pomocí spouštění událostí na základě uživatelské akce.

Jako každý moderní framework i Tapestry strukturuje souborový systém aplikace podle potřeb, vyčleňuje komponenty, třídy a stránky a to pomocí adresářů a balíčků.

Session (přihlášení uživatelé, předávání hodnot mezi požadavky) jsou řízeny pomocí aplikačních stavů.

6.1 Pracovní prostředí

Abychom úspěšně nasadili Tapestry 5 framework do IDE (nejčastěji NetBeans nebo Eclipse), musíme splnit několik podmínek. Za prvé mít nainstalované JDK a Tomcat servlet kontejner. Některé vývojové prostředí jej mají přímo zabudovaný, takže není nutná instalace kontejneru. Dalším užitečným nástrojem je Maven pro tvorbu projektového managementu. Před samotnou instalaci všech potřebných verzí prostředí a nástrojů, je více než žádoucí zjistit si kompatibilní verze na stránkách projektu[1].

6.1.1 Systémová nastavení

Po nainstalování všech stažených podpůrných softwarů nyní musíme nastavit proměnné prostředí `JAVA_HOME` k **jdk** uložišti a `PATH` k **jdk /bin** a Maven /**bin** složce v souborovém systému. U práce jsem použil tyto distibuce a verze aplikací:

- Apache Maven – 2. 2. 1
- Java version: 1. 6. 0_17
- NetBeans 6. 8 se zabudovaným Apache Tomcat 6. 0. 20
- Java EE 5

6.1.2 Vytvoření nového projektu s Apache Maven

Tento nástroj dokáže velmi ulehčit úvodní budování struktury aplikace, vytváří základní třídy, šablony a závislosti, tvoří adresářovou kostru, importuje knihovny atd. Pak už stačí v příslušném IDE vytvořit novou webovou aplikaci z existujícího zdroje a můžeme začít s implementací. Projekt přes Maven vytvoříme z příkazové řádky příkazem:

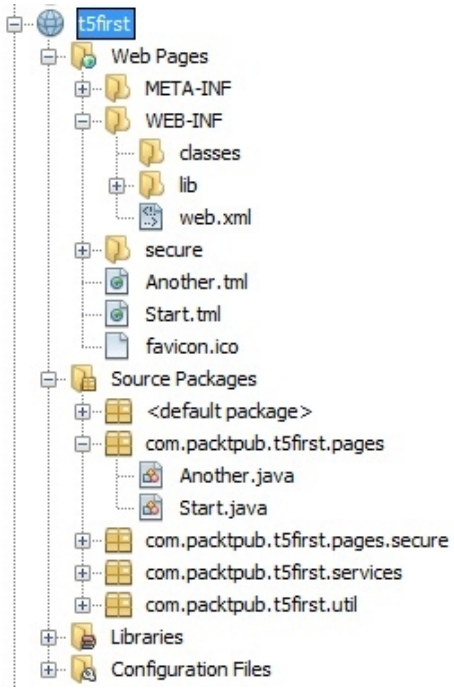
```
mvn archetype:generate -DarchetypeCatalog=http://cesta/repositar
```

Tento postup nám také umožňuje definovat název aplikace, názvy balíčků, kde se budou nacházet třídy a verzi použitého „SNAP-SHOTu“.

1 http://www.tapestry.apache.org

6.2 Struktura projektu Tapestry

Nyní máme již vytvořený a načtený projekt v našem IDE. Struktura našeho projektu je přehledně vidět na Obr. 6. 1.



Obr. 6.1: Adresářová struktura

Tapestry vyhledává stránky a komponenty pod kořenovým adresářem definovaným v minulém kroku. Java třídy jsou uloženy v **Source packages** složce stejně jako třídy komponent. Ostatní komponenty aplikace, to znamená stránky a šablony se nacházejí pod **Web Pages** složkou. V adresáři **WEB-INF**/**classes** se nacházejí zkompilované třídy. V podadresáři /**lib** jsou uloženy knihovny.

Dále se v **WEB-INF** nachází soubor **web.xml** (deployment descriptor – česky: „ popis nasazení“), v kterém je definován kořenový balíček aplikace (Obr. 6. 2) jako `<context-param>`. **Web**.**xml** Slouží obecně k specifikaci informací v servletovém kontejneru o webové aplikaci (přiřazuje servlety k URL vzorcům, jak směrovat požadavky do servletů). Avšak v Tapestry 5 je použit filtr, který je svázán s URL řetězci podobně jako servlety. Vypadá z vnějšku jako komplikovaný filtr, ale přijímá všechny požadavky a pracuje s nima.

V našem konkrétním případě podle Obr. 6. 2 budou uloženy všechy třídy stránek v `formos.com.sample_app.web.pages`. Podobně to je s komponentními třídami, které se budou nacházet v balíčku `formos.com.sample_app.web.components`.

6.2.1 Směrování požadavků

Tapestry filtr porovnává všechny příchozí požadavky a pokud splňují podmínky putují do Tapestry, zbytek je směrován do servlet kontejneru.

Aktuální soubory uvnitř webové aplikace mají přednost v případě neshody v jmenné konvenci před Tapestry stránkami.

Tapestry také rozpozná kořenové URL aplikace kde je cesta serveltů prosté „/“ a zobrazí aplikační stránku „index“, pokud je přístupná.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
        PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>sample_app Tapestry 5 Application</display-name>
    <context-param>
        <param-name>tapestry.app-package</param-name>
        <param-value>formos.com.sample_app.web</param-value>
    </context-param>
    <filter>
        <filter-name>app</filter-name>
        <filter-class>org.apache.tapestry5.TapestryFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>app</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

*Obr. 6.2: Soubor **web.xml***

6.2.2 Konfigurace projektu

Většina ostatních nastavení se nachází uvnitř třídy aplikačního modulu. Zde se definují nové služby, přepisují se zde stávající nebo se vytváří rozšíření k nastavení služeb.

Služba je užitečný kód, který vstupuje do aplikace ze vzdáleného místa přes rozhraní. Nemůže být přímo v aplikaci změněna (její zdrojový kód), zato může být ovlivněno její chování [14]. Tapestry přímo nabízí a staví na používání služeb, především vlastních. Tyto lze po implementaci injektovat do tříd a tím využít jejich funkcionalitu.

Tapestry vyhledává aplikační modul v servisním balíčku pod kořenovým adresářem. Kapitalizuje název filtru a přidá výraz „Module“. Podle Obr. 6. 2. by se konkrétně jednalo o `formos.com.sample_app.web.services.AppModule`.

Pokud taková třída existuje je přidána do registrů IoC. Aplikační modul poskytuje možnosti nastavení a přepsání implicitních symbolů definovaných Tapestry. Symboly jsou prostředky, které umožňují nastavit některé zabudované služby v aplikaci. Mohou být přepsány rozšířením AppliacationDeafult konfigurace služeb. Symboly jsou vždy definovány jako řetězce znaků, spojených s příslušným datovým typem. Některé z nich také mají spojení v konstantách třídy SymbolConstants. Následující příklad názorně ukazuje, jak vypadá předefinování některých služeb.

- Metoda v třídě AppModule.java:

```
public static void contributeApplicationDefaults(
        MappedConfiguration<String, String> configuration)
  {
    configuration.add(SymbolConstants.SUPPORTED_LOCALES, "en,cs");
    configuration.add(SymbolConstants.PRODUCTION_MODE, "false");
  }
```

Tabulku s určitými vybranými službami lze nalézt v příloze A. (Tab. č. 1)

6.3 Zpracování požadavku

Porozumět zpracování toku požadavků je důležité, jelikož se jedná o jedno z hlavních rozšíření v Tapestry. Některé počáteční fáze zpracování požadavků se provádí pomocí rozšířitelných *pipelines*.

6.3.1 Tapestry Filtr

Jak jsem zmínil v přechozím bodě všechny příchozí požadavky jsou filtorvány přes Tapestry filtr nastavený v **web.xml** souboru. Filtr se stará o spouštění a inicializování mnoha funkcí v aplikaci. Jakmile obdrží požadavek, zavolá si službu HttpServletRequestHandler a zavolá metodu `service()`.

6.3.2 HttpServletRequestHandler Pipeline

Pipelines inicializují počáteční zpracování požadavků. Rozšíření se uskutečňuje navázáním HttpServletRequestFilteru do servisní konfigurace HttpServletRequestHandleru.

Dalším pojmem jsou tzv. *terminators,* kteří buďto ukládají požadavek a odpověď do služby **RequestGlobal**, což je služba vláken, která do jednoho vlákna uloží práve jeden požadavek. Nebo zabalí požadavek a odpověď jako **Request** and **Response** a pošle je do **RequestHandler** *pipeline*.

6.3.3 RequestHandler Pipeline

Tato *pipeline* je místem, kde lze nejvíce využít rozšíření spojené s požadavky. Požadavek zastupuje abstrakci nad HttpServletRequest třídou. Tento krok je nezbytný pro neservletové aplikace. Z objektů požadavků nebo odpovědí získávají služby nebo metody v Tapestry přístup k informacím, např. dotazovacím parametrům.

Některé filtry jsou v pipeline již zabudované:

- *CheckForUpdate* – Načítání tříd a šablon
- *Localization* – identifikace uživatelského lokálního nastavení
- *StaticFiles* – ověřuje, zda-li URL patří statickým souborům. (Je předán servletu a uskutečněn přímo)
- *ErrorFilter* – zachytává neodchycené vyjímky z nižších úrovní Tapestry.

Jakmile projde požadavek přes filtry dále do aplikace, následuje *MasterDispatcher*, což je služba, která ošetřuje pokročilé Tapestry požadavky.

6.3.4 Služba Master Dispatcher

Služba *MasterDispatcher* je vlastně řetězec příkazů spojující několik objektů Dispatcher, kde každý pracuje s jinou formou URL.

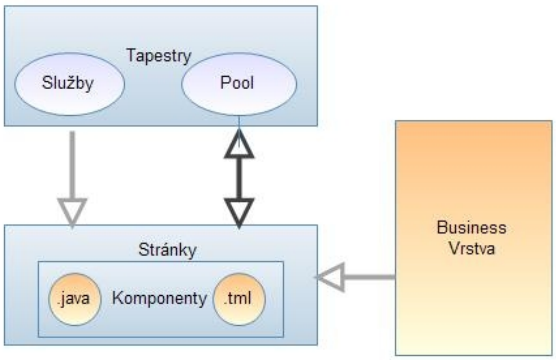
Pro podrobnější rozbor slouží komplexní obrázek: Obr. A. 1 v příloze A. Ačkoliv vyhodnocování požadavků v Tapestry je velmi komplexní, vzhledem k průchodům a předávání požadavků v *handlerech* a filterech v *pipeline,* právě díky tomu, že je Tapestry velmi přátelské k vývojáři, nemusí mít prográmátor nejmenší tušení o tom, jak se vlastně požadavek zpracovává a může dál používat události na velmi vysoké úrovni programování. Nemusí řešit dilemata, kdy

požadavek přemostit do jiného *handleru* nebo kdy použít daný *terminator*. Tapestry jednoduše zpracuje požadavky podle již implementovaných schémat zpracování. A vyhodnocování příchozích požadavků. Jakmile příjde požadavek v View spustí se naprogramovaná událost v Controlleru a Tapestry se již postará, pomocí procesů běžících v pozadí, o její správné vykonání.

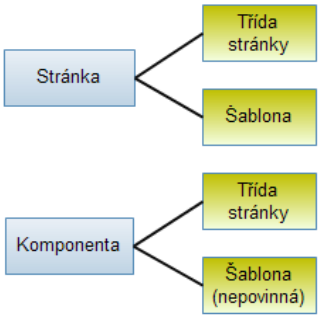
6.4 Komponenty, stránky a šablony

Každá stránka v Tapestry má svou Java třídu uloženou ve zdrojových balíčcích a jméno stránky vychází z jména Java třídy (jmenují se stejně a to včetně velkých písmen). Každá stránka má svou šablonu s koncovkou *.tml* – Tapestry Markup Language a definuje, jaké komponenty se v dané stránce nalézají. Některé komponenty mohou mít svoje vlastní šablony anebo obsahovat další komponenty. Vzniká takhle hiearchická struktura komponent.

Spojení mezi daty, Tapestry a stránkami je demonstrováno na Obr. 6. 3. Stránky figurují jako spojová vrstva mezi Tapestry a datovou vrstvou. Tapestry manipuluje s komponentami, injektuje do nich svoje služby, ze který čerpají (např. služby datového spojení s datovou vrstvou) a pracují s nimi. Tapestry inicializuje stránky přes svůj *pool*. K němu se dostanu v další kapitole.



Obr. 6.3: Struktura Tapestry aplikace



Obr. 6.4: Složení komponent

V Obr. 6. 4 je rozdíl mezi stránkou a komponentou. Stránka musí být tvořena, alespoň základním HTML nebo XML tagy (tzv. šablonou). Na rozdíl od stránky může být komponenta tvořena pouze Java třídou. Nicméně komponenta je v podstatě stránka.

6.4.1 Stránky

Problémem webovým aplikací je způsob, jak oddělit prezentační část od Modelové logiky. Toto téma jsme již rozebrali v předchozích kapitolách, kde figurovalo řešení zavedení aplikační architektury MVC. Tapestry řeší tento problém zavedením stránek, která se skládají ze tří částí a to šablonové třídy – výše diskutovaný *.tml* soubor, pak *.java* třídy a *.properties* soubor, který obsahuje popisy a nadpisy pro danou stránku, případně její jazykovou variaci. Spojení těchto tří částí vytvoří komponentu.

6.4.2 Třída Java

Každá třída může mimo jiné obsahovat vlastnosti, zachytávače nebo služby.

Získání vlasnosti z třídy lze docílit dvěmi způsoby (Obr. 6.5):

- Označením annotací: `@Property`
- Pomocí definování vlasnostni a implementací „getteru“ a „setteru“

Obr. 6.5: Vlastnosti: Property vs. Getter/Setter

6.4.3 Šablony

Soubory Tapestry šablon jsou *well-formed* XML dokumenty s *.tml* koncovkou. Název šablony musí přesně souhlasit s jménem třídy. Templaty jsou ukládány v adresáři Web Pages anebo přímo v kořenových balíčcích, pokud jsou součástí definované komponenty.

6.4.3.1 Jmenný prostor

Tapestry používá při definování komponent *namespace* definovaný v úvodním HTML tagu. (Obr. 6. 6)

```
<html xmlns:t="http://tapestry.apache.org/schema/tapestry_5_1_0.xsd"
xmlns:p="tapestry:parameter">
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:t="http://tapestry.apache.org/schema/tapestry_5_1_0.xsd"
      xmlns:p="tapestry:parameter">
    <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
        <title>${title}</title>
    </head>
```

Obr. 6.6: Hlavní tag šablony s daným jmenným prostorem Tapestry a parametrem

Všechny elementy, které se nenacházejí v jmenném protoru tapestry, budou poslány na klientskou stranu beze změn.

6.4.3.2 Expanze

Nám dovolují číst vlastnosti příslušných stránek nebo komponent. Začínají '${' sekvencí a končí '}'. Můžeme je použít v běžném textu nebo v hodnotách atributů. I u expanze můžeme použít svazovací prefixy.

6.4.3.3 Výrazy vlastností

Výrazy jsou vždy vyhodnocovány v kontextu stránky nebo ve svých komponentách. Použití operátorů '.' a '?' dovolí navigaci výrazů k objektům a vlastnostem a mohou být volány kdykoliv.

```
<t:output t:value="${user.birthDate}">
<p> ${time.hashCode()}</p>
```

Část třídy k této šabloně s expanzemi, by vypadala následnovně.

```
Private Date time;
@Property
private User user;
```

Výrazy vlastností jsou aktualizovatelné pouze pokud ve finální podobě definované jako vlastnosti a mají v příslušné třídě vytvořen „setter“ nebo jsou označeny jako *@Property*. Zde nezávisí na malých či velkých literách. (Příloha A, Tab. č. 10)

6.4.3.4 Vázací prefixy

Implicitní vázací prefixy pro většinu komponentních parametrů jsou: **prop:**, což znamená výraz dané vlastnosti: `t:label="prop:theLabel"` v šabloně a v třídě pomocí `getTheLabel()` metody. V některých případech může mít parametr rozdílný prefix např.: **literal:**. (Příloha A, Tab. č. 11)

Na Obr. 6. 7 jsou dva druhy použití vázacích prefixů. První má barvu oranžovou, je to defaultní prefix označující v komponentě název titulku. Díky anotacím se stal požadovaným a slouží navíc jako parametr v každé stránce.

```
defaultAtribut = BindingConstant.Literal
```

Atribut znamená, že se jedná o řetězec, který se nachází v hlavním `<html>` tagu každé stránky a je definován následovně: `t:title="titulek"`.

Další vázací prefixy jsou umístěny v expanzích (modrá a zelená). Klíčové slovo `message:` značí, že se bude daný výraz za dvojtečkou nacházet v nějakém *message catalogu* v aplikaci. Prvně prohledá soubory se stejným jménem jako daná šablona tzn. **Layout**.**properties** a pokud jej najde nebo jeho jazykovou variaci (**Layout_en**.**properties** v případě, že lokální jazyk je nastaven na „en“), pak podle klíčového slova (např. *userName*) přiřadí z .**properties** souboru dané klíčové slovo – *User-Name.*

6.4.4 Komponenty

Tapestry je „widgetový“ nebo-li komponentně založený framework. To znamená, že se skládá z částí kódu zvaný komponenta. Každá komponenta se dá popsat jiným způsobem a vlastnostmi. Tapestry obsahuje komponenty základní, předdefinované a také nabízí možnost tvorby vlastních. Každá komponenta má svoje *id* a *type*. Typ určuje, kterou instanci třídy Java přiřadit. Id je zase unikátní identifikátor v komponentním kontejneru všech prvků v aplikaci.

6.4.4.1 Přidávání komponent do šablony

Tapestry komponenty se objevují v šablonách jako prvky uvnitř jmeného prostoru. Kromě definice ve formuláři se mohou natypovat i mimo něj, však stále by měly být umístěny v Tapestry jmenném prostoru.

```
<t:pageLink t:page="secured/EditGoods"> !Create new! </t:pageLink>
```

Jméno elementu je komponentní typ a shoduje se s komponentní třídou. Parametry jsou vázany s použitím atributů prvků. Každé komponentě lze přiřadit unikátní id pomocí `t:id` atributu. Takto označovat komponenty se vyplatí hlavně ve formulářích. Komponentní id se totiž mohou objevit v URL nebo v klientském JavaScriptu.

Na obr. 6. 7 jsou očividné vazby mezi šablonou (**Layout.tml**), třídou (**Layout.java**) a message catalogem (**Layout_en.properties**). Tyto tři soubory tvoří celou komponentu „Layout“, která slouží jako podklad ke všem ostatním stránkám a komponentám. Lze tedy říct, že stránka je vlastně komponenta.

Další zajímavou částí je `<t:form t:if=”loginForm”>` a označuje předefinovanou Tapestry komponentu. Id určuje unikátní jméno komponenty. V třídě je pak použit prefix pro označení *event handleru* patřícímu k schválení právě tohoto formuláře. Formulář obsahuje pak další Tapestry komponenty identifikované přes vlastní *t:id* a definované pomocí *t:type.*

Obr. 6.7: Vázací prefixy, komponentní Id a type, klíče message catalogu.

6.4.4.2 Předdefinované komponenty

Tapestry obsahuje sadu předdefinovaných komponent, které se dají okamžitě použít v aplikaci s plnou funkcionalitou.

- *Linkové komponenty* – Tyto komponenty směrují na další stránky nebo spouštějí definované události. (Příloha A, Tab. č. 2)
- *Dynamické výstupní komponenty* – Komponenty, kde vyrenderování obsahu závisí na na daném mechanismu. Například na vstupní kolekci anebo třeba splnění podmínky bloku if. (Příloha A, Tab. č. 3)
- *Formulářové kontrolní komponenty* – Komponenty, jejichž funkcionalitu lze využít nejlépe při použití formulářů. (Příloha A, Tab. č. 6)
- *JavaBean komponenty* - Tapestry využívá i JavaBean komponenty, které jsou jednoduše sestavitelné a nabízejí velké využití. (Příloha A, Tab. č. 4)
- *Komponenty pracující s AJAXem* – Komponenty využívající a pracující s AJAXem. (Příloha A, Tab. č. 5)

Výčet těchto Tapestry předdefinovaných komponent není úplný. S každou updatovaou verzí přichází nové komponenty, popřípadně jejich aktualizace. Dají se také najít další komponenty vytvořené třetí stranou. Jak bylo zmíněno Tapestry je open source framework, takže obsáhnout všechny komponenty a změny v jejich parametrech je prakticky nemožné. Doporučuji vyhledat aktuální updaty na Apache Tapestry stránkách[1].

1 http://tapestry.apache.org/tapestry5.1/tapestry-core/ref/

6.5 Anotace

Tapestry používá anotace ke změně chování metod a polí vlastností komponentních tříd aplikace. Většina annotací definovaných v modulech *tapestry-core* a *tapestry-ioc* jsou velmi specifické k Tapestry komponentám nebo IoC službám. Některé anotace jsou určené k označení komponent cizích, jako: *Grid* a *BeanEditForm* a dovolují vytvářet společně vylepšení v UI bez dalšího kódování. Nejvíce se jich využije v Modelu, kde mohou anotace sloužit k označení vlastností, které se nemají v šabloně objevit, například *@NonVisual* nebo příkaz k validaci vlastnosti na úrovni UI: *@Validate*. Takovou separací dosáhneme rozdělení aplikace v ohledu na MVC architektonický vzor mnohem snadněji.

- *Anotace polí* – Rozšiřuje možnosti a chování polí viz. Obr. 6. 5 je ukázka anotace pole. (Příloha A, Tab. č. 7)
- *Třídní anotace* – Jedná se o označení třídy, pokud má rozšiřovat svou funkčnost o JavaScriptové knihy nebo explicitní *stylesheet*. (Příloha A, Tab. č. 8)
- *Anotace metod* – Užívá se například pokud se jedná o událostní metody (Obr. 6. 8). (Příloha A, Tab. č. 9)

6.6 Události komponent

Tapestry pracuje s aplikačním kódem čtením a aktualizací vlastností a zpracováváním vyvolaných událostí. Tyto metody zpracovávání událostí mohou být uživateli přístupné či nepřístupné. Komponentní události mohou být spuštěny z klientské strany např. schválením formuláře anebo na straně serveru či kombinací obojího.

Každá událost má své jméno, nejčastěji se jedná o označení „action“. Některé komponentí spouštěče spouštějí jiné události. Každá událost může mít svůj kontext, který do URL přiřadí nějakou hodnotu komponentní vlastnosti jako řětězec typu *String*. Tyto kontextní hodnoty jsou poté převedeny zpátky do patřičného datové typu a vystupují jako parametr v událostní metodě.

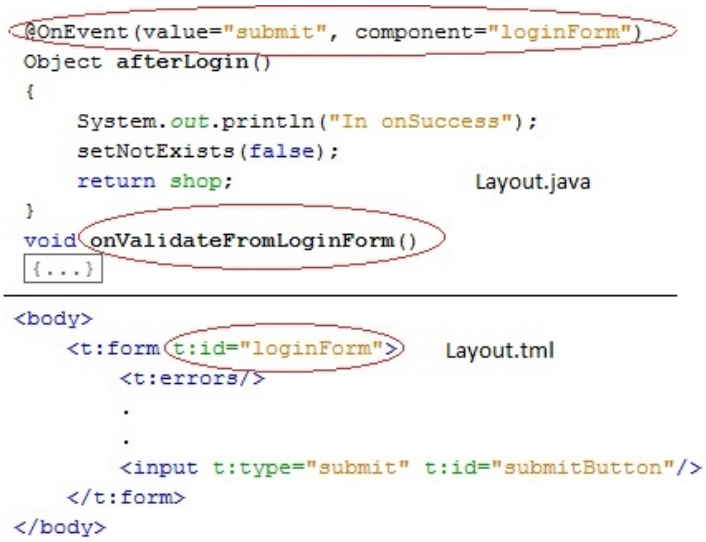
6.6.1 Jmenná konvence událostí

Metody událostí mhou nabývat podoby `onEventName` anebo `onEventNameFromComponentId`. Na obr. 6. 8. lze vidět spojitost mezi označením metody k id komponenty.

6.6.2 Anotace událostí

Anotace @OnEvent je variací k jmenné konvenci událostí. Srovnání na obr. 6. 8.

```
@OnEvent(value="submit",component="loginForm")
void afterLogin(){}
```



Obr. 6.8: Značení událostních metod pomocí annotací

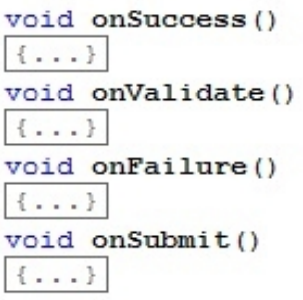
6.6.3 Návratové hodnoty

Návratové hodnoty z událostních metod slouží stejně jako v jiných frameworcích (např. JSF) k navigaci mezi stránkami. (Příloha A, Tab. č. 12) V první metodě je návrácen objekt *shop*. Pokud jsme do třídy injektovali instanci třídy Shop, pak bude vše v pořádku. Příklad injekce zdroje stránky:

```
@InjectPage
private Shop shop;
```

6.6.4 Formulářové události

Komponenty formulářů vyvolávají řadu událostí při zobrazování a při potvrzování formuláře. (Příloha A, Tab. č. 13)



Obr. 6.9: Vyhodnocovaní formuláře

Všechny tyto události se vyhodnocují postupně. Deklarace a funkcionalita těchto metod je v režii programátora. Mohou obsahovat validční kód nebo v případě *failure*, přesměrování na jinou stránku, atd. (Obr. 6. 9.)

6.7 Lokalizace

Lokalizace v Tapestry je založena na zjištění, jaké je uživatelovo nastavení lokálního jazyku a to buď z HTTP hlavičky anebo z přítomnosti HTTP *cookie* v uživatelově webovém prohlížeči.

Každá stránka a komponenta může obsahovat svůj vlastní *message catalog* (katalog zpráv), který obsahuje klíčové názvy hodnot a k ní přiřazené hodnoty v daném jazyce. Několik takových souborů, tvoří aplikační *message catalog*. Pokud je povolených více lokálních jazyků, každý takový soubor se pak jmenuje stejně akorát s přidaným sufixem se zkratkou daného jazyka. Pokud je implicitním nastavením soubor: **MojeStranka1.properties**, pak její jazyková variace třeba v angličtině bude vypadat takto: **MojeStranka1_en.properties**. U těchto souborů platí, že podřízený *message catalog* k dané komponentě přepisuje klíče z nadřízeného katalogu. Nejvyšší *message catalog* je **app.properties**.

Všechny tyto zprávové katalogy s předefinovanými klíčemi jsou dostupné ve stránce přes expanze s vázaným prefixem: `${message:vzkaz}, ${message:vitejte}`

Komponentní *message kalatogy* mohou být také injektovány do tříd jako objekty typu *Message*. Za použití anotace `@Inject`. Toto platí i o injekci stránkových lokálních variací `java.util.Locale`.

```
@Inject
private Message message;
@Inject
private Locale suppertedLocale;
```

6.8 Live class and template reloading

V překladu okamžité znovu načtení tříd a šablon. Nová vlastnost Tapestry 5 je automatické znovu-načtení změněných tříd a šablon. V předchozích verzích a jiných frameworcích vyžaduje opětovné načtení tříd restartování Servlet kontejneru nebo znovunasazení celé webové aplikace. Třídy stránek a komponent jsou automaticky načteny po provedení změn. Stejné je to s komponentními šablonami a přidruženými zdroji.

6.8.1 Template reloading

Při změně šablony se všechny instance stránky (spolu s podřízenými komponentami, které se na této stránce nachází) zahodí a znovusestavení probíhá s novou šablonou. V tomto případě se znovunačtení netýká tříd těchto instancí.

6.8.2 Class reloading

Při změně jakékoliv z tříd načtených z kontrolního balíčku nebo jeho pod-balíčku, Tapestry zahodí všechny stránkové instance a třídní *loader*.

Tento krok neovlivní persistentní datová pole, protože jsou uložena odděleně v session. Tento postup dovoluje provádět poměrně zásadní změny v komponentních třídách, aniž by přerušily běh aplikace.

Pouze třídy těch stránek a komponent, které definované v kořenovém balíčku, mohou být

načítány. O který balík se jedná si Tapestry vyhledá v souboru **web.xml**.

6.9 Tapestry služby

Jak již bylo řečeno Tapestry nabízí ruzné užitečné služby, které mohou být injektovány do komponent nebo jiných služeb. Některé služby, které nejsou vkládány do cizích subjektů, však mohou být ručně nastaveny a využity v aplikaci. (Příloha A, Tab. č. 14)

6.10 Shrnutí

Nastínil jsem zde hlavní funkcionalitu a vlastnosti Tapestry, jak se vyrovnat s klasickými implementačními paradigmaty webových aplikací. Tapetry nabízí všechny podstatné základy každého webového frameworku a ty složitější implementuje svým způsobem. Ve většině případů se jedná o návrh řešení problematiky, ke kterému se přistupuje ze získaných zkušeností za léta vývoje. Nicméně i zde se některé paradigmaty řeší vlastní implementací řešení a někdy bohužel složitějším procesem.

7. Tapestry5 aplikace

Aplikace, kterou jsem při své práci vyvíjel nastíňuje základní charakteristiky frameworku. V této kapitole naznačím řešení některých problému webových aplikací v Tapestry 5. Jednodušší postupy, jak definovat komponenty a psát tagy v šablonách byly probrány v předchozí kapitole. Zde se budu zabývat zajímavými řešeními jiné problematiky.

Zde uvedu hlavní implementace problémů navigace mezi stránkami, jak mezi jednotlivými požadavky předávat a uchovávat hodnoty polí tříd a z toho plynoucí možnosti záložkování. Dále vysvětlím, jak Tapestry řeší uchovávání údajů o uživateli v session pomocí objektu aplikačního stavu. Tapestry také poskytuje pěkné řešení pro hlášení chyb. Dále jak může být pohodlné a jednoduché vytvořit dvě nejzákladnější a přitom komplikované komponenty (*Grid* a *BeanEditform*), pomocí několika řádků kódu. Následuje validace formulářů a polí a nakonec jak funguje vzor MVC v Tapestry.

7.1 Navigace

Po vyhodnocení požadavku ze strany uživatele mu aplikace poskytne webovou stránku podle jeho požadavků. Uživatel učiní další požadavek a přesune se na jinou stránku. Tapestry ošetřuje tento chod událostí pomocí svého mechanismu. Postupně je uživateli poslána první stránka a poté je tato vrácena do tzv. *poolu,* kde je vyčištěna od vložených dat a připravena k dalšímu použití. Jakkékoliv hodnoty do ní vložené jsou ztraceny. Tapestry tímto způsobem ušetří mnoho prostředků z toků dat (zde vydíme využití podobné tzv. *cachování*), jelikož stránky nejsou nikde vytvářeny či diskratovány, jednoduše se využijí znova při požadavku jiného klienta. To má za následek, že se hodnoty neudrží mezi požadavky jednotlivého uživatele a musí se ošetřit mechanismus k jejich uložení. Tapestry využívá dvě možnosti, jak se vypořádat s tímto problémem.

7.1.1 Persistentní stránkové pole

Zde se nabízí možnost použití anotace `@Persist` k označení pole v třídě, které tak získá schopnost uchovat svou hodnotu než bude znova použito.

```
@Persist

private String pole;
```

Při persistentním označením pole Tapestry vytvoří HTTP *session* pro daného uživatele a informaci uloží na server. Tento scénář se může stát zátěží u aplikací, kde se pohybuje více uživatelů a tím pádem větší požadavky na prostředky ze serveru. Navíc tento postup neumožňuje přidat stránku do záložek.

7.1.2 Page activation context

Další možnost jak nezatížit server a předat hodnotu mezi stránkami je stránkový aktivační kontext. Zde figurují dvě Tapestry metody, které učiní stránku aktivní. To znamená, že v určitém okamžiku životního cyklu požadavku Tapestry přesměruje toky hodnot z HTTPRequestu do aktivního místa ve stránkách.

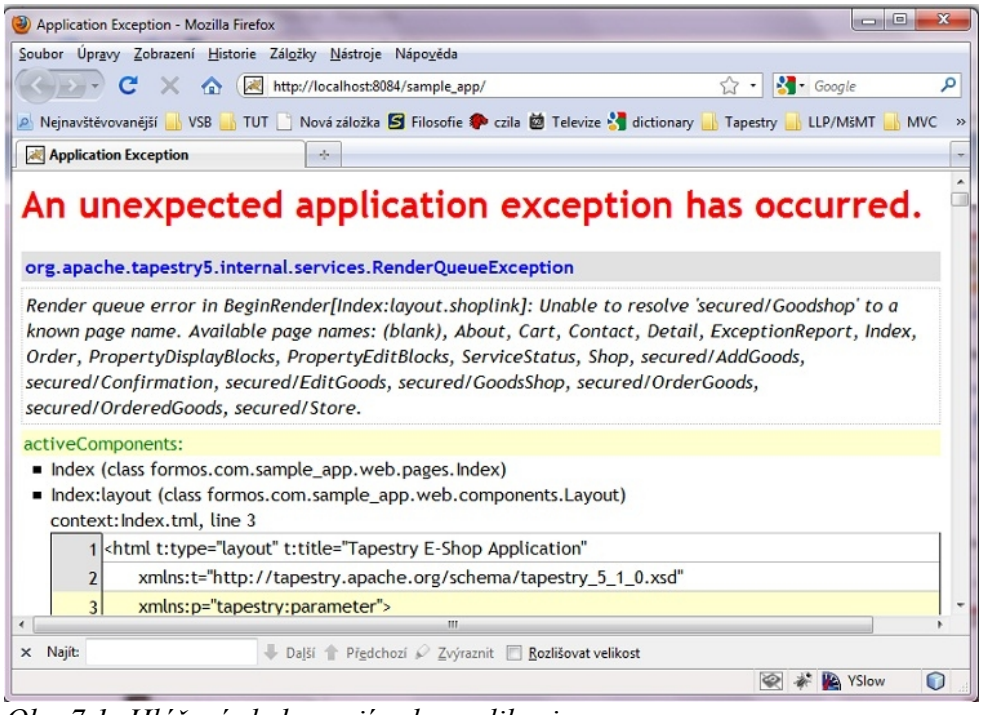
```
void onActivate(String name){
```

```
        if(name.length()>1){
            goods = goodsDAO.getGoodsByName(name);
            this.goodsName = name;
        }
    }
    String onPassivate(){
        return goodsName;
}
```

Metoda `onPassivate` způsobí, že se před vrácením předchozí stránky do *poolu*, aktuální stránka „zeptá“ předchozí, zda-li zde není nějaké kontext stránky, který má být uchován. Pak si je převezme a vloží do příslušného pole aktuální stránky metodou `onActivate`. Navíc se tento paramter zobrazí v URL, takže lze z něj vytvořit záložku, což je někdy velice přínosný funkce aplikace. URL, na které se pak daný objekt odkazuje vypadá třeba takto: **http://localhost:8084/sample_app/cs/secured/editgoods/Black$0020Chocolate**

7.2 Hlášení chyb

Tapestry má velice pěknou a užitečnou nástavbu. Jedná se o zobrazování chyb ve webovém prohlížeči. Kromě výpisu standartních vyjímek apod., vypisuje i informace o chybě v templatu a také samozřejmě barevně označený řádek v kódu a třídě, kde se chyba vyskytla. Navíc poskytuje nápovědu možností, jak daný problém vyřešit.



Obr. 7.1: Hlášení chyb a vyjímek v aplikaci

7.3 Application state object, konfigurace,

Objekt, který je v Tapestry přístupný všem stránkám se nazýva *Appication State Object.* Představme si, že aplikace nabízí možnost přihlášení, pak se samozřejmostí uchovat si data přihlášeného uživatele po určitou dobu nebo dokud se neodhlásí. Právě proto v Tapestry existuje tento druh objektu, který je vlastně *session* aplikace.

```
@ApplicationState
private User user;
public User getUser()
{
      return user;
}
```

Tato definice vytvoří v *session* aplikace objekt User, který je pak přístupný pro celou aplikaci. Při vytvoření dalšího `ApplicationState` objektu třídy User se bude pořád jednat o tu samou instanci vytvořenou prvně.

Důležitou informací o ApplicationState objektu je fakt, zda-li již byl vytvořen. Znalost exixstence tohoto objektu je důležitá, pokud naše aplikace pracuje s *loggováním*. K tomu nám postačí nové pole definované v třídě dané stránky s názvem objektu a sufixem: `exists`, datový typ `boolean` a jeho `get` metoda.

```
private boolean userExists;
public boolean getUserExists()
{
      return userExists;
}
```

V případě potřeby dodání datových zdrojů do užívaných stránek, můžeme využít Application state object (ASO) uplně stejně jako v přechozím případě. Takže řekněme, že existuje v naší aplikaci modelová třída, která se stará o business logiku dat (tvoří obraz tabulky namapované z databáze). Pak je zde rozhraní přes které využíváme dané metody a funkce. Následné vložení rozhraní do stránek vypadá takto:

```
@ApplicationState
private IGoodsSource goodsDAO;
```

Nyní by vše fungovalo, jenomže ještě musíme přidat právě definici rozhraní do ASO konfigurace a **AppModule.java**, aby Tapestry dokázal spojit rozhraní s danou třídou a navíc vytvořil správné spojení do *session*.

Metoda, která bude vložena do této třídy, rozšiřuje služby poskytované Tapestry.

```
 public void contributeApplicationStateManager(
              MappedConfiguration<Class, ApplicationStateContribution>
configuration){ApplicationStateCreator<IGoodsSource> creator =
              new ApplicationStateCreator<IGoodsSource>(){
          public IGoodsSource create(){
              return new FalseGoodsSource();}
      };configuration.add(
                                                IGoodsSource.class,   new
ApplicationStateContribution("session",creator));}
```

V těle metody se nachází definování nové spolupracující služby a poté přidání této definice v metodě `configuration.add()`.

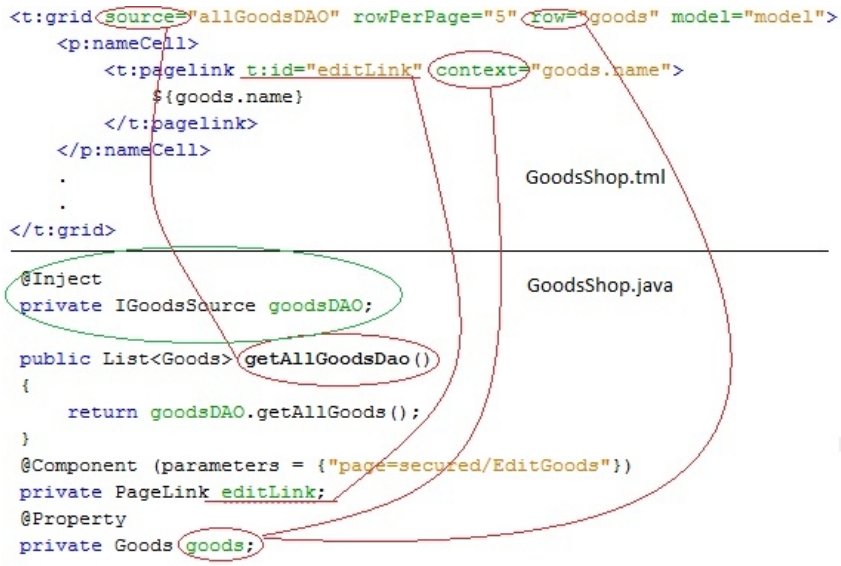
Definice nových a rozšiřování služeb není lehkou záležitostí, vyžaduje jisté znalosti vnitřní funkcionality frameworku, které by se do této práce nevešlo.

V nové verzi Tapestry 5.1 se vyskytuje použití označení pro ASO `@SessionState`.

7.4 Komponenty Grid a BeanEditForm

Tapestry nabízí řadu zajímavých komponent, které ulehčují programátorovi práci.

Grid ve velice mocný nástroj, jelikož stačí určit zdroj dat a komponenta si již sama zpracuje všechny atributy a hodnoty a navíc nabídne možnosti třízení nebo stránkování. Vybere první objekt z kolekce a zjistí jeho vlastnosti, podle kterých vytvoří atributy tabulky (obr. 7. 2). *Source* je propojen s metodou, která vrací kolekci objektů získanou z injektovaného rozhraní `IGoodsSource`. Atribut *row* pak definuje podle jakého pole se má tabulka listovat.



Obr. 7.2: Komponenta Grid a šablona

Dále vidíme na obrázku v atributu tabulky možnost odkázání se na objekt, který má hodnotu vybraného paramteru objektu `Goods`. Z Obr. 7. 3 se přes stránkový odkaz a s ním vybraný kontext můžeme přesunout na další stránku, kde se nám naskýtá možnost editace vybraného objektu. Kontext je přenesen přes vlastnost objektu `Goods.name` na další stránku pomocí aktivního stránového kontextu metod `onActivate` a `onPassivate`.

Obr. 7.3: Náhled Grid komponenty v prohlížeči

BeanEditForm je zobrazen na obrázku 7. 4. Pro definici všech parametrů, potřebných k jejich zobrazení nám postačí pouze instance třídy `Goods` a `get` metoda.

```
private Goods goods;
public Goods getGoods() {
      return goods; }
```

Obr. 7.4: BeanEditForm - zobrazení

Na komponentě vidíme, že nám nabízí textová pole a také správně definované hodnoty ze získaného objektu. Navíc je zde velmi pěkně použit JavaScriptový *datePicker*. Není nutná žádná další implementace kromě aplikační logiky po stisknutí tlačítka. V šabloně stačí napsat:

```
<t:beaneditform t:id="form" t:object="goods" t:submitLabel="Update">
        </t:beaneditform>
```

`t:object` definuje z jakého objektu třídy má komponenta čerpat data. Další pěknou vlastností je automatická validace *BeanEditFormu*. Pokud zadáte jiný datový typ hodnoty do pole s celými čísly například, automaticky se objeví jednoduchá klientská validace.

7.5 Validace

Správná validace v moderních aplikacích by měla:

- Upozornit uživatele, že některá vstupní hodnota byla chybně zadána.
- Chybné pole by mělo být nějakým způsobem označeno.

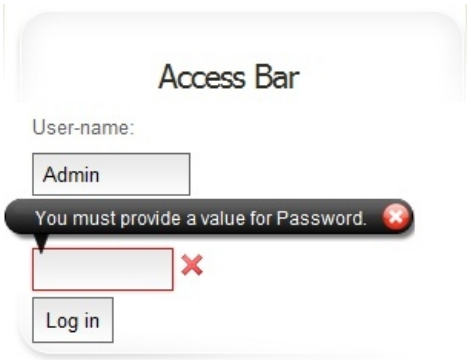
- V lepším případě zobrazit špatně zadanou hodnotu a vysvětlit, proč nastala chyba.

Validace se aktivuje buďto vložením tzv. validátorů do šablony a jejich aktivaci pomocí Tapestry komponenty `<t:errors/>`.

```
<t:form t:id="registrationForm">
<t:errors/>
      <input type="text" t:type="textfield" t:id="userName"
      t:validate="required,minlength=3,maxlength=8"/>
```

Nebo přidáním anotace v poli třídy. Tento druh lze použít v třídách, které zastupují Model a umožňují tak uchování business logiky v Modelu (Obr 7. 5).

```
@Validate("required")
private String userName;
```



Obr. 7.5: Indikace chyby z šablony

Validace probíhá tak, že se prvně na klientské straně figuruje JavaScript a pomocí validátorů označuje chybná pole nebo se hodnoty zpracují překladačem a pošlou na server, kde je validátor vyhodnotí. (Obr. 7. 6)

Obr. 7.6: Chyba na serverove straně

Validace v Tapestry nabízí možnosti validování regulérních výrazů, délek řetězců apod.

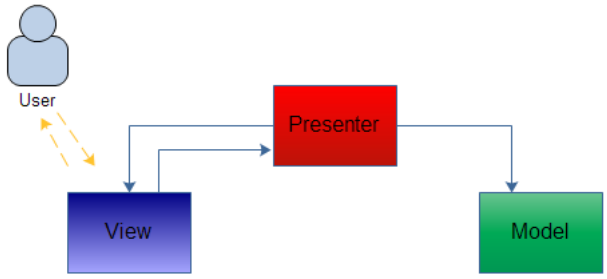
Také pracuje s *ValidatorTracker* třídou, která umožňuje rozšířit možnosti validace ve formulářích. Hlášky, které se objevují ve výstupních hlášeních, pak lze pomocí *message catalogů* měnit podle potřeby.

7.6 Implementace MVC

Jak Tapestry implementuje MVC architektonický vzor? Jak se v Tapestry rozdělují jednotlivé vrstvy a co obsahují?

Shrněme si postup, jak se v aplikaci projeví uživatelský požadavek. Uživatel vstoupí na stránku a požádá a zobrazení dat. Požadavek je poslán přes View do třídy stránky, která sloužila jako UI. V třídě - (Controlleru) se aktivuje událost, která zpracuje požadavek a podle něj zaktivuje metody v Modelu, aby provedly operaci nad daty, změnily uložená data atd. Třída poté pošle do View požadované hodnoty a ten je zobrazí uživateli. Controller zde zastává funkci jak spouštěče procesů v Modelu, tak i prostředníka, který data pošle zpět do View.

Z tohoto pohledu je patrné, že View zde zastává funkci zobrazovacího prostředku. Controller zase řídí všechny toky událostí, jak z View, tak i z Modelu. Proto bych se přiklonil k definici o widgetových komponentních architekturách, kde se definuje tento vzor jako Model – View – Presenter. A v tomto případě jeho variace ***Passivate View***.



Obr. 7.7: Vzor Passive view

- View – šalblony stránek a komponent (.tml)
- Presenter – Java třídy stránek a komponent (. java)
- Model – Business logika, spojená datová rozhraní se svými modelovými třídami.

  Třídy modelu se do Presenteru vkládají pomocí služeb Tapestry (Injekcí a ASO objekty).

8. Závěr

Při tvorbě Tapestry 5 byl brán zřetel na problematiku vývoje webových aplikací komplexněji než u jiných frameworků. Nejednalo se o výstavbu aplikace na „zeleném stole“, ale o obecnější náhled vycházející ze zkušeností, omylů, chyb a testování. Aktuální verze Tapestry je 5 z čehož vyplývá, že před ní byly k dispozici jiné a mnohem méně sofistikované systémy. Tapestry je ve vývoji již po dobu 10-ti let.

Taprestry nabízí řešení všech funkčních požadavků moderních webových aplikací. Některé implementuje pomocí klasických vzorců, s jinými si poradí svým způsobem anebo využitím cizího systému jako frameworku Hibernate nebo Spring. Tapestry se snaží vytvořit prostředí, které je přívětivé programátorům. Hledá řešení problematiky v nejjednodušších postupech, které dovolují v návrhu aplikace méně kompromisů vůči běžnému uživateli.

V porovnání s jinými frameworky je konfigurace mezi JSP a servlety velmi snadná. Oproti Struts a JSF stačí jednoduchá jmenná konvence a filtr požadavků, namísto přiřazování vyhodnocování požadavků mezi JSP a servlety pro každou událost.

Tapestry také velmi názorně rozděluje aplikaci do architektonických vrstech vzoru MVP a používá komponentní systém. Je přehledný a jednoduchý. Třídy jsou čisté POJO objekty s importovanými knihovnami funkcí. Šablony jsou *well-formed* XML dokumenty a ve struktuře aplikace je vše velmi pěkně uspořádáno do logických celků.

Tapestry využívá všech metod pro aplikaci moderních požadavků jako SEO (pomocí URL rewritting), AJAX (knihovny), znovupoužitelnost komponent a validaci na všech vrstvách. Neopomenu lokalizaci a multijazyčnost za využití *message catalogů*.

Vlastní IoC kontejner dovoluje aplikaci členit do malých částí, které pak můžou být použity jako služby a injektovány na požadovaná místa podle aktuálních potřeb aplikace. Toho řešení není příliš jednoduché, ale zato nabízí nepřeberné množství možností.

Skutečně objektově – orientovaný framework. Tak se píše v knize Alexandra Kolesnikova. Vytváření objektů v *session* pomocí ApplicationState anotace je toho důkazem. To samé se týká injekce stránek jako objektů do jiných tříd.

Framework Tapestry 5 byl pro mě prvním systémem pro tvorbu aplikací, s kterým jsem se seznámil velmi dobře. Programování a vůbec chápání celé jeho funkcionality pro mě bylo jednodušší než u jiných frameworků. Nebo spíše – nenastal problém, kterých bych nebyl schopen vyřešit pomocí dostupné dokumentace a zdrojů.

A zde bych rád vytkl několik věcí. Velmi malá dostupnost informací. Tapestry není příliš rozšířený mezi „javisty“, a proto také menší komunita, znamená menší přístup k novým publikacím. To je škoda a doufám, že se to v budoucnou zlepší, jelikož i Tapestry 5 je oproti starších verzím sofistikovanějším nástrojem pro tvorbu webových aplikací.

Ačkoliv nejsem znalcem Java frameworků, Tapestry se nestalo problémem, jak vytvořit aplikaci. Problém byl pochopit funkce Tapestry, abych mohl jednoduše navrhnout přehlednou aplikaci. I přesto, že snadné řešení není věrohodné, doporučuji vyzkoušet. Genialita přece spočívá v jednoduchosti.

Literatura

Seznam použité literatury.

[1] http://en.wikipedia.org/

[2] http://cs.wikipedia.org/

[3]Avgeriou, Paris; Uwe Zdun (2005). "Architectural patterns revisited:a pattern language". *10th European Conference on Pattern Languages of Programs (EuroPlop 2005), Irsee, Germany, July*.

[4] DVOŘÁK, Miloš. *Návrhové vzory (design paterns)* [online]. 2003 [cit. 2010-04-14]. Dostupné z WWW: <http://objekty.vse.cz/Objekty/Vzory>.

[5] Erich Gamma, Richard Helm, Ralph Johnson, John Vlisside: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, říjen 1995, ISBN 0201633612

[6]The Open Group Architecture Framework . *The Open Group Architecture Framework (TOGAF 8.1.1 'The Book')* [online]. [s.l.] : Van Haren Publishing, 2007 [cit. 2010-04-13]. Architecture Patterns, s. . Dostupné z WWW: <http://www.opengroup.org/architecture/togaf8-doc/arch/chap28.html>. ISBN 9789087530945.

[7] TICHÝ, Jan. *Funkční požadavky webových aplikací*. Praha, 2004. 66 s. Diplomová práce. Vysoká škola ekonomická v Praze.

[8] BERNARD, Bořek. MVC a další prezentační vzory. *Root.cz* [online]. 2009, č.1, [cit. 2010-04-14]. Dostupný z WWW: <http://zdrojak.root.cz/clanky/uvod-do-architektury-mvc>.

[9] APPLETON, Brad. *Http://www.cmcrossroads.com/* [online]. 2000 [cit. 2010-04-15]. Patterns and Software: Essential Concepts and Terminology. Dostupné z WWW: <http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html>.

[10] HRADECKÝ, Radim. *Srovnání softwarových framework·*. Praha, 2009. 89 s. Diplomová práce. ČVUT v Praze.

[11] *Docforge.com* [online]. 2010 [cit. 2010-04-15]. Web application framework. Dostupné z WWW: <http://docforge.com/wiki/Web_application_framework>.

[12] KOLESNIKOV, Alexander . *Tapestry 5 Building Web Application*. Birmingham - Mumbai : PACKT - Publishing, 2007. 275 s.

[13] Apache Software Foundation. *Tapestry* [online]. 2006 - 2010 [cit. 2010-04-27]. Dostupné z WWW: <http://tapestry.apache.org/>.

[14] FOWLER, Martin. *Inversion of Control Containers and the Dependency Injection pattern* [online]. 2004 [cit. 2010-05-03]. Inversion of Control Containers and the Dependency Injection pattern. Dostupné z WWW: <http://www.martinfowler.com/articles/injection.html>.

Seznam příloh

Příloha A

Nachází se zde tabulky a obrázky ke kapitole 6.

Příloha B

Návod k nasazení aplikace do NetBeans 6. 8 IDE.

Příloha A

V této příloze se nachází tabulky z kapitoly 6, které obsahují popis základních komponent, annotací, formulářových událostí, návratové hodnoty událostí, atd.

Tapestry symboly

Pole SymbolConstants	Popis	Defaultní hodnota
CHARSET	Výstup a požadované kódování	UTF-8
COMPRESS_WHITESPACE	Pokud je „true“, tak odstraní nadbytečné prázdné místa v šabloně. „False“ je zanechá.	TRUE
PRODUCTION_MODE	„true“ pro zkrácené výpisy vyjímek. „false“ pro plný výpis vyjímek	TRUE
SUPPORTED_LOCALES	Čárkou oddělený seznam jmen povolených lokálních jazyků.	En, cs, de, fr, ...

Tab. č.1 - Konstantní symboly

Předdefinované komponenty

- Linkové komponenty

Linkové komponenty	Popis
ActionLink	Spouští událost na komponentě.
EventLink	Spouští běžnou událost.
PageLink	Vytváří odkaz k poskytnutí dané stránky aplikaci.

Tab. č.2 - Linkové komponenty

- Dynamické výstupní komponenty

Dynamické výstupní komponenty	Popis
If	Renderuje aplikaci svůj blok, pokud jsou splněny podmínky.
Loop	Renderuje svůj vnitřní blok několikrát, iteruje kolekci.
Output	Formátuje na výstup objekt pomocí třídy Format.

Tab. č.3 - Dynamické výstupní komponenty

- JavaBean komponenty - Tapestry využívá i JavaBean komponenty, které jsou jednoduše sestavitelné.

Stavební komponenty	Popis
BeanDisplay	Zobrazení read-only vlastností JavaBeany.
BeamEditor	Vytvoří UI pro JavaBean. Poskytuje rozdílné typy formulářových ovladaču každého atributu.
BeanEditForm	Kombinace BeanEditoru a formuláře s potvrzovacím tlačítkem.
Grid	Výstup v podobě tabulky setu JavaBeanů se stránkováním s seřazováním.

Tab. č.4 - JavaBean komponenty

- Komponenty pracující s AJAXem

AJAXové komponenty	Popis
AjaxFormLoop	Speciální iterační komponenta využívající se uvnitř formuláře pro interaktivní práci s řádky.
FormFragment	Dovoluje měnit viditelnost části formuláře.
Zone	Přijímač dynamického kontextu ze serveru. Poskytuje okamžité aktualizace.

Tab. č.5 - AJAXové komponenty

Anotace

- Anotace polí

Anotace polí	Popis
@InjectPage	Injektuje stránku, která obsahuje tuto komponentu jako read-only pole
@Parameter	Definuje pole jako formální parametr komponenty. Pole mohou být optimální nebo vyžadované, mohou povolovat nebo zakazovat hodnotu null anebo mít přednastavenou hodnotu.
@Persist	Identifikuje pole, jejichž hodnota by měla přetrvat mezi požadavky. Je uložená v session.
@Property	Tapestry vygeneruje k poli této anotace „getter“ a „setter“.

Tab. č.7 - Anotace polí

- Formulářové kontrolní komponenty

Formulářové kontrolní komponenty	Popis
Checkbox	Přepínací tlačítko
Errors	Zobrazuje vstupní validační chyby pro daný formulář
DateField	Textové pole s JavaScriptovým vyskakovacím kalendářem
Form	Kontejner pro formulářové kontrolní komponenty
Label	Label pro formulářové komponenty
LinkSubmit	JavaScriptový odkaz k schválení formuláře
PasswordField	Textbox pro vkládání nezobrazujících se znaků
Radio	Přepínací tlačítko s jediným výběrem hodnoty
RadioGroup	Nezobrazená komponenta pro orgranizaci Radio komponent
Select	Rolovací seznam
Submit	Potvrzovací formulářové tlačítko
TextArea	Víceřádkové vstupní textové pole
TextField	Jednořádkové vstupní textové pole

Tab. č.6 - Formulářové kontrolní komponenty

- Třídní anotace

Anotace tříd	Popis
@IncludJavaScriptLibrary	Zajišťuje, že specifické JavaScriptové knihovny jsou správně nalinkovány do označeného výstupu.
@IncludeStyleSheet	Zajišťuje, že soubor daných kaskádových stylů je správně nalinkován na značený výstup.
@SupportsInformalParameters	Označí komponentu, aby podporovala dodatečné neformální parametry.

Tab. č.8 - Třídní anotace

- Anotace metod

Anotace metod	Popis
@Log	Tapestry zaloguje vstup a výstup metody na úrovni debuggování.
@OnEvent	Označí metodu ke zpracování událostí
@Cached	Návratová hodnota metody je uložena do schránky a může být později načtena bez opětovného vytvoření.
@CommitAfter	Potvrzuje transakci po provedení metody.

Tab. č.9 - Anotace metod

Výrazy vlastností

Výrazová forma	Popis
true, false	Boolean.true, Boolean.false
null	null
This	Aktuální stránka nebo komponenta.
1234, 1234.56	Číslo java.lang.Long, číslo java.lang.Double
name	Jméno vlastnosti.
metoda()	Jmeno metody, která bude volána.
Name.metoda	Vnořená vlastnost. Hodnotí vlastnost name, potom vlasnost metoda.
Name?.metoda	Bezpečná dereference: Hodnotí name pak metodu dokud není name rovno null, v tomto případě je výraz vlastnosti roven null.
'retezec'	Literální řetězec uvnitř jednoduchých apostrofů.

Tab. č.10 - Výrazy vlastností

Vázací prefixy

Základní vázací prefixy	Význam
block:	Id bloku v šabloně.
component:	Id komponentního potomka. Používá se ke spojení dvou komponent. Nejčastěji se jedná o Label a TextBox.
literal:	Literální řetězec.
message:	Číslo java.lang.Long, číslo java.lang.Double
prop:	Výraz vlastnosti.

Tab. č.11 - Vázací prefixy

Návratové hodnoty událostí

Typ	Význam
String	Jméno stránky, která bude zobrazena.
Class	Třída stránky, která bude zobrazena.
Object	Instance stránky k zobrazení. Používá se přes @InjectPage annotaci.
StreamResponse	Posílá tok bajtů na stranu klienta.
Java.net.URL	Externí URL k přesměrování.
boolean	Vrátí true, aby ukončil událostní „vybublání“.

Tab. č.12 - Návratové hodnoty událostí

Formulářové události

Jméno události	Podmínka	Použití
prepareForRender	render	Připraví stav stránky k zobrazení formuláře.
prepareForSubmit	submit	Připraví stav stránky na zpracování formulářového potvrzení.
prepare	render/submit	Připraví stav stránky k renderová a potvrzení.
validateForm	submit	Uskuteční závěrečnou validaci, jakmile jsou všechny pole zpracována.
success	submit	Je vyvolána, pokud žádné vstupní pole neobsahuje validační chybu.
failure	submit	Je vyvolána, pokud některé vstupní pole obsahuje validační chybu.
submit	submit	Je vyvolána jako poslední bez ohledu na to, zda-li některé pole invalidní.

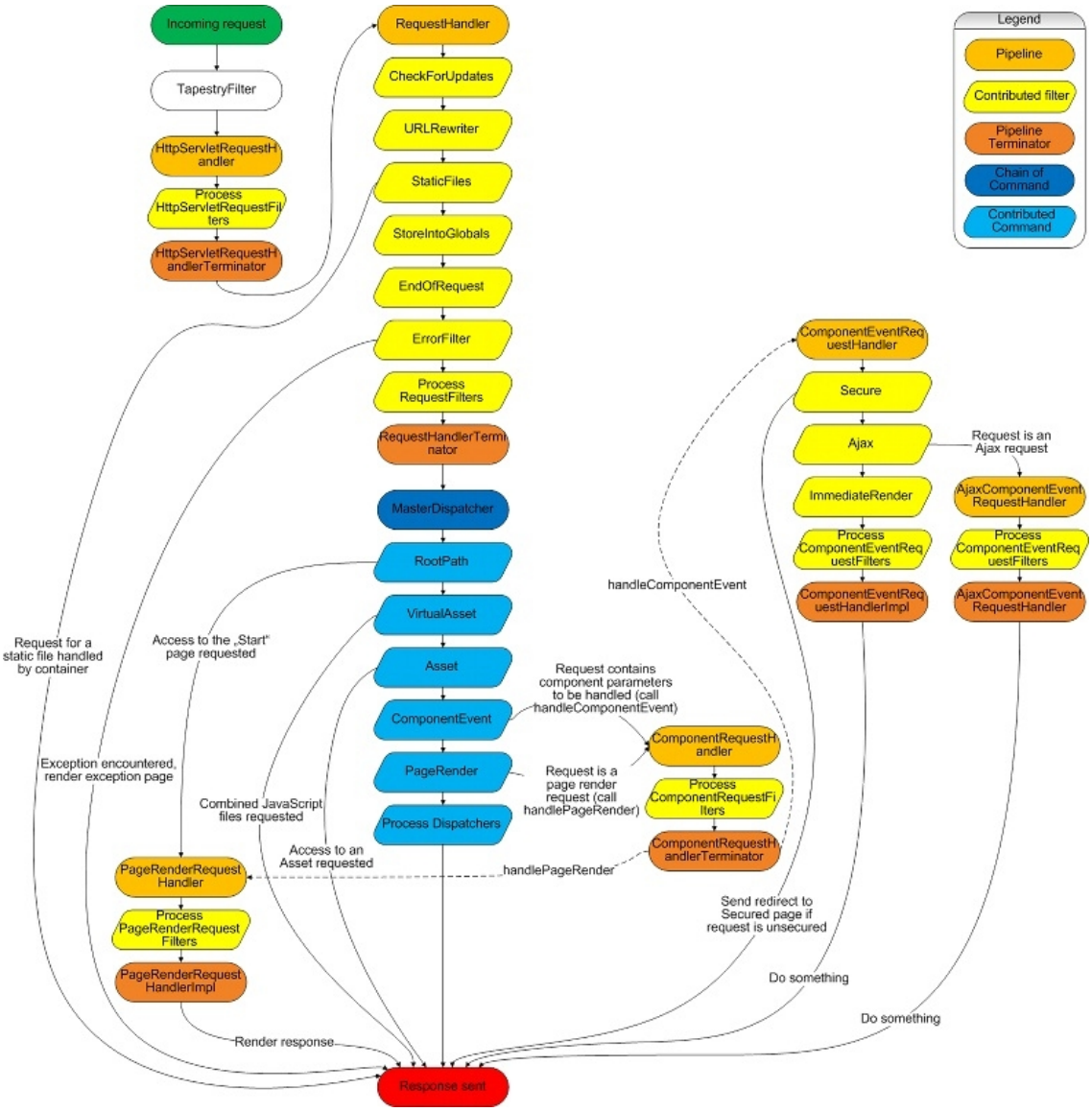
Tab. č.13 - Formulářové události

Tapestry služby

Služba	Popis
ApplicationDefaults	Konfigurace přepíšeFactoryDefaults symboly použité při konfiguraci jiných služeb.
BeanModelSource	Zdroj pro BeanEditor komponentu, která může být pozměněna podle potřeb aplikace a použita dalšími komponentami jako Grid.
ComponentSource	Poskytuje přístup ke stránkám nebo komponentám.
Request	Aktuální požadovaný objekt použitý k příménu získání hodnoty parametru.

Tab. č.14 - Tapestry služby

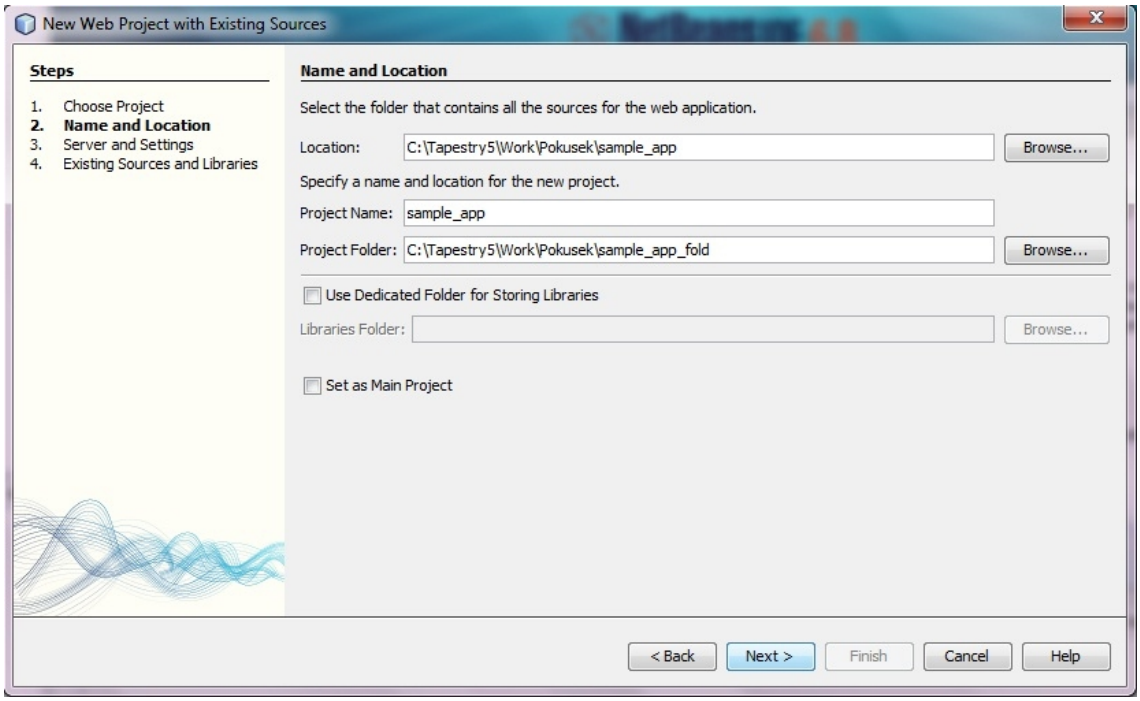
Zpracování požadavku



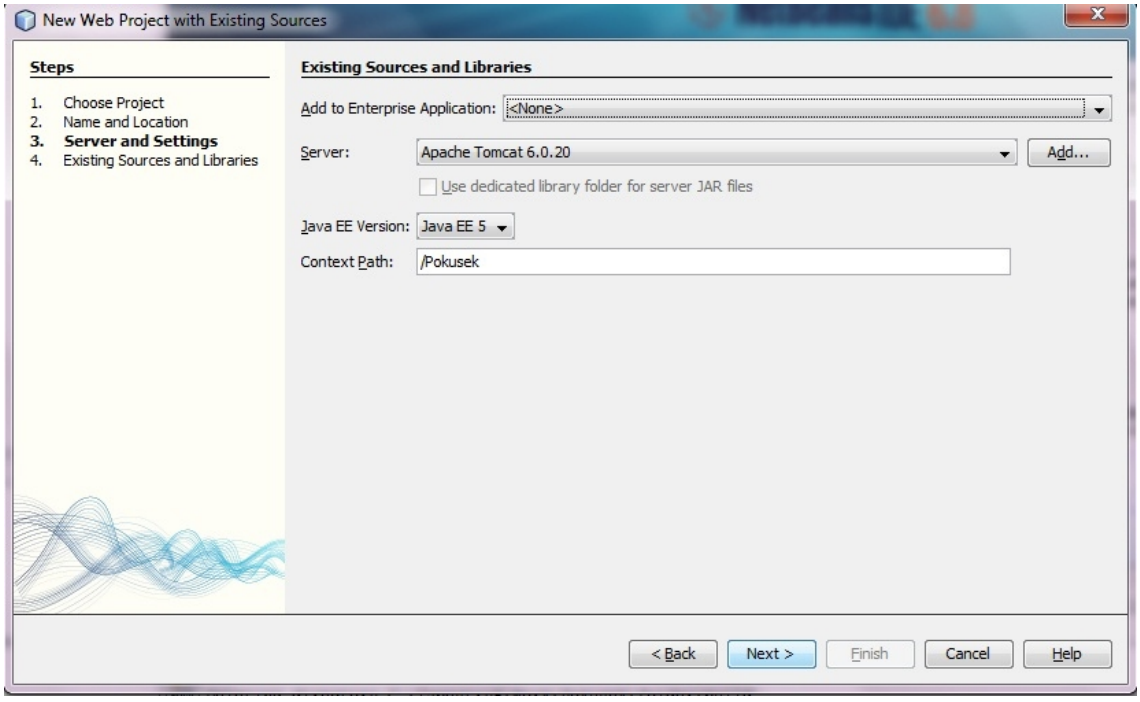
Obr. A.1: Zpracování a vyhodnocování příchozího požadavku do Tapestry aplikace

Příloha B

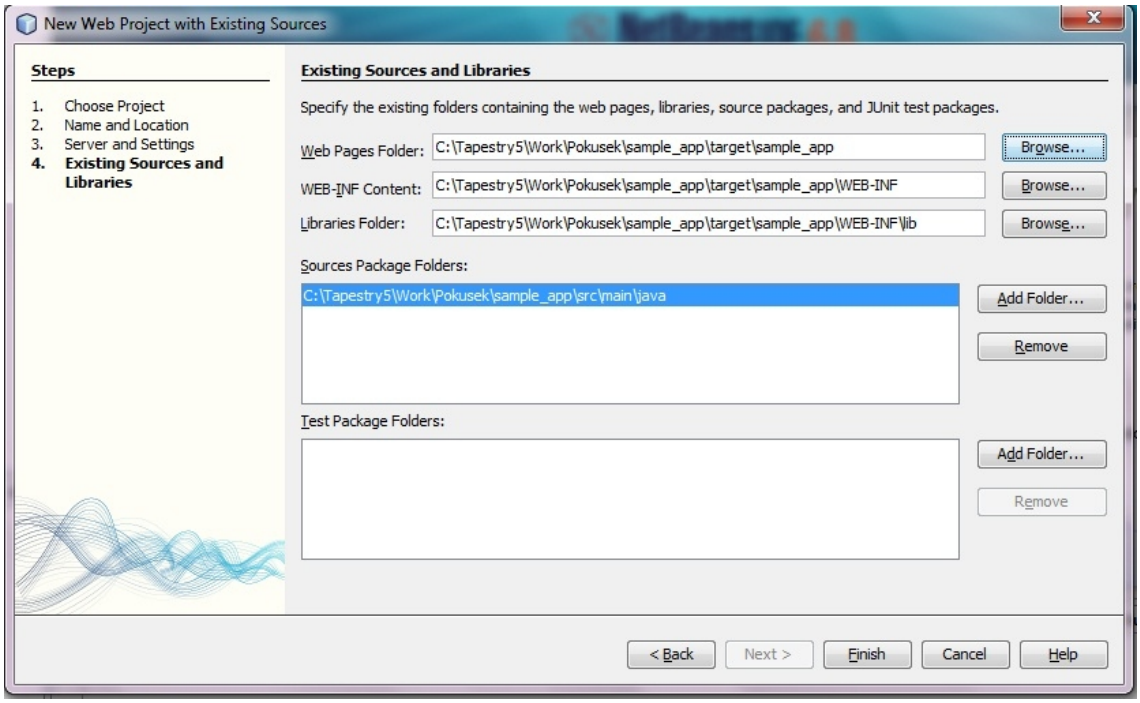
Aplikaci lze importovat jako existující web application project from existing source, stračí se řídit podle obrázků celé importace projektu. Hlavně Pozor při vybírání adresářů.



Obr. 8.1: Pojmenovani, vyber zakladniho adresare



Obr. 8.2: Výběr serveru a Verze Java EE



Obr. 8.3: Zvolení web Pages Context – zde pozor

Systém

Role / přístupy:

- Nepřihlášený uživatel – může vstoupit na stránky Domů, Info a kontakt
- Administrator – login: admin; password: admin – Můžu přidávat zvoží, editovat, zobrazovat nabídku uživatelů.
- Přihlášený uživatel – login: user; password: user – Objednává zboží do košíku, z něj jej může odebírat a provádí objednávku